Diss. ETH No. 20661

Real-Time Camera Control for Interactive 3D Applications

A dissertation submitted to
ETH Zurich

for the Degree of
Doctor of Sciences

presented by
Thomas Oskam
MSc in Computer Science, ETH Zurich, Switzerland
born 19. February 1982
citizen of the Netherlands and Switzerland

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner
Prof. Dr. Matthias Zwicker, co-examiner
Dr. Alexander Hornung, co-examiner

2012

Abstract

Real-time 3D applications have evolved to the point where they become increasingly realistic. Several aspects have been subject to extensive research in order to deliver the maximum amount of realism possible within a limited computation budged. Examples are rendering or physics. One aspect, however, has not gotten the attention needed despite its omnipresence in any application, namely camera control and parametrization. In most interactive applications such as games, the camera placement and parametrization is either user controlled or pre-scripted by an artist. Only few attempts have been made to automate and simulate realistic camera behavior, as it is generally a difficult task. In this dissertation, we attempt to attack these shortcomings on different levels.

In the first part of this thesis we present a real-time camera control system that uses a global planning algorithm to compute large, occlusion free camera paths through complex environments. The algorithm incorporates the visibility of a focus point into the search strategy, so that a path is chosen along which the focus target will be in view.

In the second part, this thesis deals with camera parametrization for controlled stereoscopic rendering. We present an automatic controller for camera convergence and interaxial separation that specifically addresses challenges in interactive 3D applications like games. In such applications, unpredictable viewer or object motion often compromises stereopsis due to excessive binocular disparities. We derive constraints on the camera separation and convergence that enable our controller to automatically adapt to any given viewing situation and 3D scene, providing an exact mapping of the virtual content into a comfortable depth range around the display.

Finally, the third part of the thesis approaches advanced camera parametrization and the reproduction of realistic color balancing effects. The input to our algorithm is a sparse set of desired color correspondences between a source and a target image. The global color space transformation problem is then solved by computing a smooth vector field in CIE La^*b^* color space that maps the gamut of the source to that of the target. Furthermore, we show how the basic per-image matching can be robustly extended to the temporal domain. This extension renders our method extremely useful for automatic, consistent embedding of synthetic graphics in video, as required by applications such as augmented reality.

Zusammenfassung

Dreidimensionale Echtzeitanwendungen haben sich mehr und mehr zum Fotorealismus hin entwickelt. Verschiedene Teilbereiche wurden dabei ausführlich erforscht, wie zum Beispiel die physikalische Simulation oder die Bildwiedergabe. Ein Aspekt hat allerdings bisher nicht ausreichende Beachtung gefunden, obwohl seiner Allgegenwart: Kamerakontrolle und -parametrisierung. In den meisten Anwendungen wird die Kamera durch den Benutzer kontrolliert oder deren Bewegung durch Künstler vordefiniert. Es wurden nur wenige Versuche unternommen das Verhalten einer Kamera zu automatisieren. Diese Dissertation versucht diese Mängel auf verschiedenen Stufen anzugreifen.

Im ersten Teil dieser Doktorarbeit präsentireren wir ein Kontrollsystem das weite und verdeckungsfreie Kamerapfade durch globales planen in Echtzeit erreichen kann. Der Algorithmus integriert die Sichtbarkeit zu einem Fokuspunkt in die Suche damit ein Pfad gewählt wird bei dem der Fokus im Sichtfeld bleibt.

Der zweite Teil dieser Doktorarbeit behandelt die Parametrisierung der Kamera um kontrollierte stereoskopische Bilderzeugung zu erreichen. Wir präsentieren einen automatischen Regler um die Konvergenz und Separation der verwendeten Kameras zu kontrollieren, so, dass den speziellen Bedürfnissen von interaktiven 3D Applikationen Sorge getragen wird. In solchen Anwendungen, unvorhersehbare Bewegungen des Betrachters oder von Objekten können die dreidimensionale Wahrnehmung kompromittieren wenn übermässige Bilddisparitäten entstehen. Wir leiten deshalb Formeln für die Separation und Konvergenz der Kameras her, die unserem Regler erlauben, automatisch auf neue Situationen zu reagieren. Dies erlaubt uns, jede dreidimensionale Szene in einen genau definierten wahrgenommenen Tiefenbereich abzubilden.

Schliesslich, im dritten Teil dieser Arbeit, behandeln wir fortgeschrittene Parametrisierung der Kamera durch realistische Farbkorrektur von Bildern. Die Eingabe unseres Algorithmus ist eine spärlich verteilte Menge von Farbreferenzen zwischen zwei Bildern. Die globale Farbraumtransformation wird durch das berechnen eines glatten Vektorfeldes im CIE La^*b^* Farbraum erreicht welches die Farbskala des Quellbildes in die des Zielbildes überführt. Desweiteren zeigen wir, wie dieser Ansatz robust auf die Zeitachse erweitert werden kann. Diese Erweiterung erlaubt das automatische und konsistente Einbetten von künstichem Bildmaterial in ein Video, welches von Anwendungen wie Augrentierte Realität vorausgesetzt wird.

Acknowledgments

My cardinal thanks go to my advisor Prof. Markus Gross. His ideas and insights inspired me to follow my passion and empowered me to finish this thesis. I sincirely thank Dr. Alexander Hornung, who has proven to be an excellent Supervisor and was able to push my work to new levels. I also want to thank Dr. Robert W. Sumner for his many ideas and his help. Special thanks go to Prof. Matthias Zwicker for agreeing to co-examine this thesis.

Many thanks go to all my additional collaborators that contributed in various ways to this thesis: Nils Thuerey, Huw Bowles, Kenny Mitchell, Serkan Bozyigit, Michael Spreng, Theodor Mader, Martin Banks, Peter Kaufmann, Aljoscha Smolic, Alex Stuard, and Wojciech Jarosz.

I also would like to thank my colleagues and friends from CGL, CVG, DRZ, IGL, and Blackrock. Special thanks go to Tobias Pfaff, my long time office mate, without whom the work would have felt twice as hard.

Finally, I want to thank Jan, Susy, Didier, Jeannette, Lia, and especially Linda, for their support and for believing in me.

I dedicate my dissertation to Max and Cornelia.

Contents

CHAPTER 1

Introduction

Camera control in interactive virtual environments is a fundamental concept and its proper execution an important factor in delivering a realistic experience to the user. Where the geometry in the virtual environment mirrors the complexity of the real world, and lighting computations the appearance, the camera provides the interface to the user. It is the final puzzle piece in the rendering pipeline that decides how the user is looking into the virtual world.

Similar to cameras in the real world, there are many factors of a virtual camera that influence the final appearance of the scene. Complex interaction with the environment on one side, and the user on the other side, pose difficult problems on different control levels. For example, there is a lengthy planning phase before shooting a feature film in which each scene is carefully composed. A director decides before hand where each camera is placed, how it is supposed to move, and on what it is focussed. This kind of planning is usually infeasible in virtual environments, where a user is controlling the camera. Even more difficult is the case, where the camera has to be controlled automatically, in order to free the user's interaction for other tasks. Another problem is the proper simulation of realistic camera behavior. Digital cameras have become sophisticated devices with a series of on-board image processing and color balancing during capture to enhance image quality. This kind of processing has shifted the viewer's judgement of realistic video footage. A

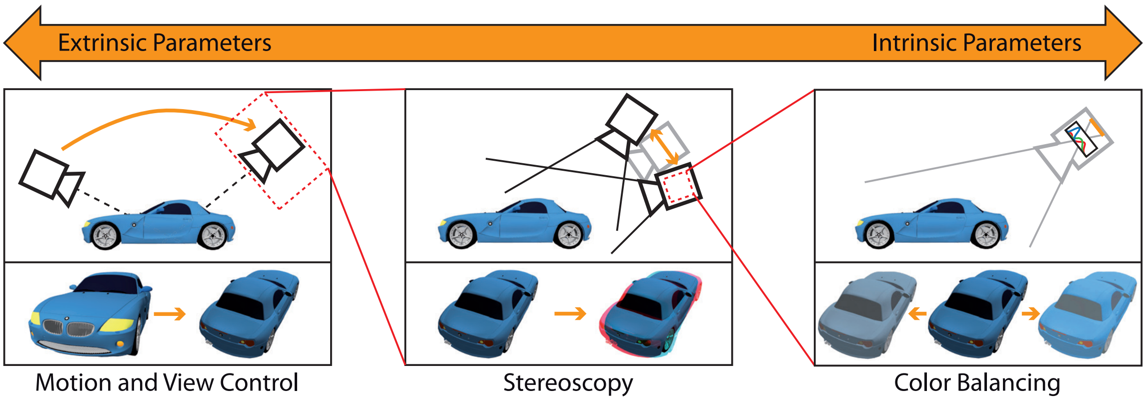


Figure 1: *Camera behavior in virtual environments can be broken down into a hierarchy of different levels, ranging from purely extrinsic, over mixed, to purely intrinsic parameter control.*

hand-held camera that is not adjusting its brightness, exposure, or contrast when filming a scene is regarded as unrealistic. While the movement of a camera is mirrored by the positioning of the camera in the virtual scene, color balancing effects such as adjusting contrast are more difficult to achieve.

In general, the problem of virtual camera control can be broken down into different levels. With applications ranging from purely extrinsic, over mixed, to purely intrinsic control, the complex nature of real cameras can be mapped to concepts in virtual environments. This is demonstrated in Figure 1. At the highest level, the camera is seen as an abstract object. At this level, only extrinsic camera parameters are considered, and it is only important where the camera is placed, what it is looking at, and how it is moved to different places. Once the camera has been positioned and oriented, mixed extrinsic and intrinsic applications appear. At this level, the cameras principal position and view direction have been defined, but the parametrization of the scene's projection still needs to be determined. Adjusting parameters such as the focal length or the opening angle of the camera control the projection of the environment. Several concepts at this level require the additional adjustment of some of the camera's extrinsic parameters as well. Examples of such concepts are the simulation of camera wiggling or the rendering of stereoscopic image pairs. Therefore, at this level, extrinsic and intrinsic parameters are controlled together to finalize the projection of the environment's geometry onto the image plane of the camera. Finally, at the finest level, the purely intrinsic camera behavior is covered. At this level, only the parameters are considered that influence the appearance of the rendered image. Concepts such as lens distortions, color shifts, or image noise are investigated here.

In this dissertation we are exemplary investigating a specific problem from each of the three levels of camera control specifically tailored to real-time scenarios in interactive 3D applications: environment-aware camera planning, optimized stereoscopic rendering, and adaptive color balancing. These three research areas are discussed in individual chapters, where emphasis is given to fast algorithmic solutions and parallelizability in order to meet the strict budges of high-performance interactive applications such as computer games.

Structure of the Thesis and Contributions

First, the basic camera pipeline in interactive computer graphics is reviewed in Chapter 2. In Chapter 3 we investigate the problem of camera view and motion planning in virtual environments. We show, how the virtual environment can be modeled using a specialized data structure. Using this data structure we are able to perform high-level camera planning tasks, such as visibility aware object avoidance or occlusion risk prediction at runtime. We propose a variety of search strategies that provide solutions to advanced camera motion planning problems in interactive environments. The results and contributions of these studies are published in Proceedings of the Symposium on Computer Animation 2009 [Oskam et al., 2009].

Mixed extrinsic and intrinsic camera behavior is then investigated in Chapter 4 on the example of automatic stereoscopic rendering. We discuss an in-depth geometric pipeline from the virtual scene to the viewer's eyes in the real world, and present a temporally smooth and automatic solution for controlling the perceived depth for arbitrary combinations of camera movements and 3D environments. Our solution is able to robustly automate the control of perceived stereoscopic depth to improve the comfort of the viewer as well as provide novel tools for improving and facilitating the production of stereoscopic content. Our optimized stereoscopic camera controller is published in Proceedings of SIGGRAPH Asia 2011 [Oskam et al., 2011].

Finally, Chapter 5 focusses on intrinsic camera behavior. In this chapter we investigate the efficient reproduction of unknown color transfer functions of digital cameras. We propose a framework that learns from example images recorded with a real camera and is able to apply these functions to new images at real-time. Furthermore, we extend the approach to the temporal domain to allow image augmentation and show how this can be employed to improve the visual realism of augmented reality type applications. The findings of these studies are published in Proceedings of 3DIMPVT 2012 [Oskam et al., 2012].

CHAPTER 2

Virtual Camera Pipeline

Generally, an interactive 3D application is executed through a main program loop. This loop first executes the computations necessary to simulate and update the current state of the virtual environment. Second, the loop performs the rendering steps that show the current state of the environment on the display. While the simulation step is usually only performed on the CPU, the rendering is also done on the GPU.

The importance of camera control has lead to the seamless embedding of the camera pipeline into this application loop. Some parts of the camera pipeline are executed on the CPU, while others are performed on the GPU. As the GPU hardware has become very flexible and programmable, it is hard to exactly draw a line at which point the camera pipeline is executed on the GPU. However, in general, extrinsic parameters tend to be easier to implement on the CPU, while the GPU usually is more suited for intrinsic parameters.

A schematic overview of the program loop for a virtual environment is shown in Figure 1. It consists of three main parts. First, the simulation of the virtual world is performed. At this stage, tasks are completed that change the state of the environment, such as animating characters, simulating physics, processing user input, or updating the internal logic of the program. Second, the camera pipeline is traversed. This stage encompasses the determination of camera position and orientation, the projection of the environment onto the camera image plane, and the final rendering of the environment. Finally,

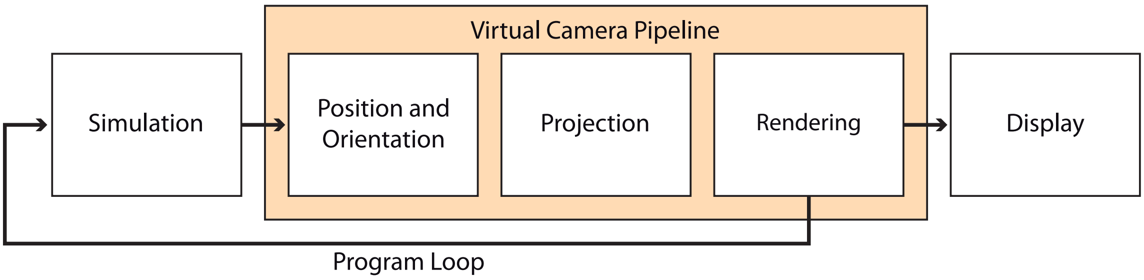


Figure 1: *Schematic overview of the program loop of a virtual environment. First, the virtual world is simulated, driving animations and the program logic one time step forward. Second, the virtual camera is controlled. Its configuration is determined, placing it into the environment. Then, the projection is determined that maps the thee dimensional environment onto the two dimensional image plane before the rendering simulates the capture of light rays by the camera from the environment. The resulting image is shown to the viewer on a display, while the program loop starts over.*

while the program loop is restarted, the rendered image is shown to the user on a display, effectively interfacing the real world with the virtual world.

In the scope of this thesis, we assume both the simulation and display stages of the program loop as given black-box processes, and focus on the camera pipeline part. In the following we will briefly revisit the individual parts of this pipeline. A complete in-depth discussion on this topic is provided by Christie et al. [2009].

1 Position and Orientation

The configuration of the camera in three dimensional space is uniquely defined by its position and orientation. While the position is always given as a three dimensional vector $C = (c_x, c_y, c_z)$, there are several formats used for the orientation. The most common way is to define the the viewing direction by the point $P = (p_x, p_y, p_z)$ the camera is looking at and the up vector $U = (u_x, u_y, u_z)$ of the virtual environment. Other common ways to formulate the orientation of a camera are Quaternions [Shoemake, 1985] or Givens rotations (also known as yaw-pitch-roll) [Bindel et al., 2002].

The camera position and orientation can be computed as the multiplication a translation and a rotation matrix. However, it is more common to transform the environment into the camera coordinate frame so that the camera is moved to the origin and looking along the Z-axis with the Y-axis being up. This approach facilitates the camera projection computations later on. The view matrix is therefore the inverse transformation of placing the camera into the scene and applied to the geometry of the environment. The translation matrix T is thus computed as

$$T = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

The rotation matrix R should now rotate the environment around the origin in such a way, that the camera view (Z-axis) is pointing at P. This rotation can be regarded as the change of the global coordinate system to the local camera coordinate system. Therefore, the local axes X^c, Y^c, and Z^c of the camera coordinate system are computed as

$$Z^c = \frac{P - C}{||P - C||} \tag{2}$$

$$X^c = \frac{U \times Z^c}{||U \times Z^c||} \tag{3}$$

$$Y^c = Z^c \times X^c, \tag{4}$$

and the rotation matrix R is computed as

$$T = \begin{bmatrix} x^c_x & x^c_y & x^c_z & 0 \\ y^c_x & y^c_y & y^c_z & 0 \\ z^c_x & z^c_y & z^c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

The view transformation matrix V can now be computed as

$$V = TR. \tag{6}$$

This matrix, for any given camera position and orientation, transforms the environment such that the origin can be regarded as the center of projection of the camera. The view matrix V thus implements the extrinsic properties of the camera. However, also fine-scale changes to the camera's configuration can be incorporated at this stage. As long as these changes manipulate the camera position or viewing direction they can be integrated into T and R.

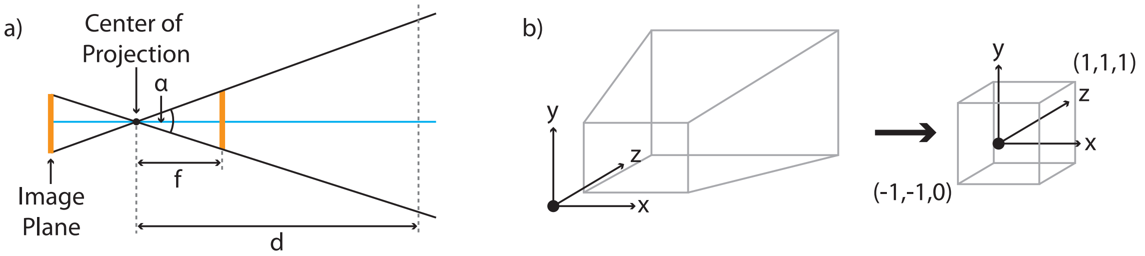


Figure 2: *Basic parametrization of a virtual camera. a) The virtual camera is defined by an opening angle α and a focal lenght f. The distance d defines the far clipping plane of the camera view frustum. b) The camera parametrization opens up a view frustum in three dimensional space. Perspective projection of the geometry is performed by transforming the view frustum into a cuboid between $(-1,-1,0)$ and $(1,1,1)$.*

There are intrinsic camera parameters, such as opening angle or image shift, that require integration into the projection part of the camera pipeline. We will discuss an appropriate camera parametrization model in the next section.

2 Camera Parametrization and Projection

To this end, parameters that correspond to the physical placement of the camera into the virtual scene have been discussed. We have shown that the camera orientation and position are inverted and applied to the scene geometry to make sure that the camera center of projection is at the origin. Now, we introduce the camera parametrization that allows to compute a projection matrix that maps the three dimensional geometry of the virtual environment onto a two dimensional image plane.

The standard parametrization of a virtual camera for interactive environments consists of the opening angle α, the focal length f, and the image aspect ratio a. Furthermore, the parameter d defines the far clipping plane. Only the geometry between f and d will be rendered. This limitation of the near and far distance is in place to allow the render depth buffer to cover a clearly defined range. A schematic overview of the camera parametrization is shown in Figure 2 a).

The camera parameters outline a view frustum in three dimensional space. The goal of the projection is to provide a transformation matrix P that, when multiplied to all the points in the scene, transforms the view frustum into a cuboid that lies within $[-1;1]$ in the X- and Y-axes and $[0;1]$ in the Z-axis. This brings the entire geometry in the scene into a form where parts outside the view frustum can be easily clipped by removing all projected geometry outside the cuboid. Figure 2 b) shows this transformation.

Given the horizontally defined opening angle α and the aspect ratio a (defined as image width divided by image height), the width w and height h of the view frustum's near plane can be computed as

$$\begin{aligned} w &= 2f \tan\left(\frac{\alpha}{2}\right) && (7) \\ h &= \frac{w}{a}. && (8) \end{aligned}$$

Now, the projection matrix P can be computed as

$$P = \begin{bmatrix} 1/w & 0 & 0 & 0 \\ 0 & 1/h & 0 & 0 \\ 0 & 0 & d/(d-f) & 1 \\ 0 & 0 & f - fd/(d-f) & 0 \end{bmatrix}. \quad (9)$$

The parameters w and h are inverted in order to make sure that the values from the X- and Y-axes are scaled according to the aspect ratio and opening angle. The value $d/(d-f)$ scales the values in the Z-axis according to the view frustums depth. Finally, $fd/(d-f)$ in the bottom row of the projection matrix translates the transformed cuboid back so that the image plane is placed on the origin.

Changes of intrinsic camera parameters such as the opening angle α, focal length f, or aspect ratio a directly influence the projection matrix P. In order to finalize the complete transformation for the virtual geometry, P can be combined with the previously derived view matrix V to form the camera transformation matrix C that transforms the entire scene into the image plane. Scene distances inside the view frustum are transformed into the scale of [0;1].

$$C = VP \quad (10)$$

3 Rendering and Advanced Effects

The rendering stage is the final part of the camera pipeline, and an entire research field on its own. Here, the geometry that has been projected into the camera system is triangulated, clipped, ordered, rasterized, and colored.

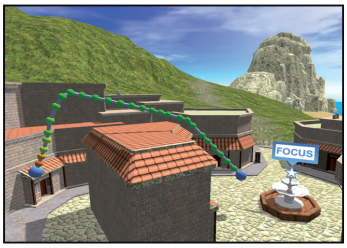
Especially the colorization step is summarizing a vast field of algorithms, as lighting is applied to the geometry and the final color of each image pixel is determined. In interactive computer graphics applications, this final stage is performed exclusively on the GPU. Through massive parallelization of the computations, sophisticated algorithms for coloring individual pixel are possible.

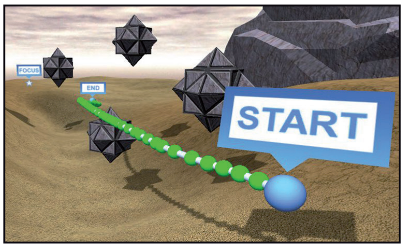
At this stage, the final polish to the finished rendering of the virtual scene is performed. It can be compared to the on-chip image processing of digital cameras during capture. After the user has pressed the trigger, the incoming light is captured and an image is formed in the camera. Then, this image information is post-processed to counter problems that arose during the capture. These are effects such as brightness adjustment, noise suppression, contrast improvement, and many more [Adams et al., ; Agarwal et al., 2006; Klein and Murray, 2010].

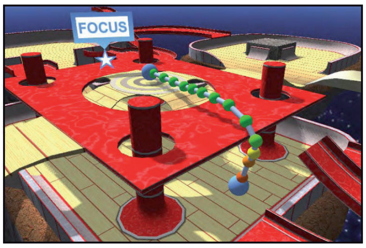
Therefore, to also finalize our hierarchical categorization of camera control, the creation of advanced color transfer functions is performed during rendering time. Thanks to the previous transformations of the scene geometry, and the following clamping and rasterization on the GPU, these type of effects can be regarded independently. Additionally, to not mess with the classical lighting, texturing, and colorization algorithms, advanced image effects can simply be implemented as another GPU shader pass at the end of the rendering.

CHAPTER 3

Dynamic Camera Motion and View Control







Moving a camera in a dynamic and unpredictable environment is a complex problem that restricts camera artists in creating an immersive experience. With our dynamic camera control, we achieve a solution that provides an entire arsenal of tools to perform automated camera control.

In any virtual environment, computer game, or other interactive application, natural camera motion is crucial for a positive user experience. Successful navigation through content strongly depends on appropriate camera movement. When considering automatic camera control, four critical criteria include:

Real-time. Static or precomputed viewpoints are unsuitable since the user's actions are not known a priori. Instead, the camera must adapt in real-time.

Collision-free. As the camera moves through a scene, it must not collide with objects in the environment as this lends an unrealistic feel and lessens the sense of immersion.

Smooth. Teleporting the viewpoint from one place to another may disorient the user since the continuity of the view is broken.

Visible. Ultimately, the camera's goal is to look at something. Thus, visibility is of utmost importance: the player or other focus target must be kept in view and unobstructed.

Classical third-person camera models that follow a player deal with these criteria using algorithms that are inherently local in nature. Small adjustments in position and orientation are made based on objects in the camera's immediate vicinity. Such local camera control is effective in some situations, but if the character dashes quickly behind corners, the locality assumptions are broken and the camera may be forced to pass through an object or teleport to an unobstructed view. Furthermore, local camera models are not designed to perform large-scale transitions between viewpoints.

We attempt to alleviate the aforementioned problems by computing smooth, collision-free transitions between arbitrary start and end points, while emphasizing the visibility of a third focus point. A typical application might be the switch from a third-person view of an avatar in a virtual city environment to an overhead view while ensuring a clear view of the avatar throughout the camera motion. Maintaining unbroken visibility to a focus point helps the user to better understand position and context. Although large-scale transitions and local camera control may seem like disparate problems, we propose that the two are closely connected. A local camera controller that deals with visibility in a sophisticated way must move the camera from one point in space to another in order to maintain or regain visibility (Figure 1). For certain configurations, this movement can only be resolved with global knowledge of the scene.

Visibility computations in arbitrary three-dimensional environments are generally complicated and time-consuming. The difficulty of this problem is exacerbated by several factors. The global nature of the transition planning

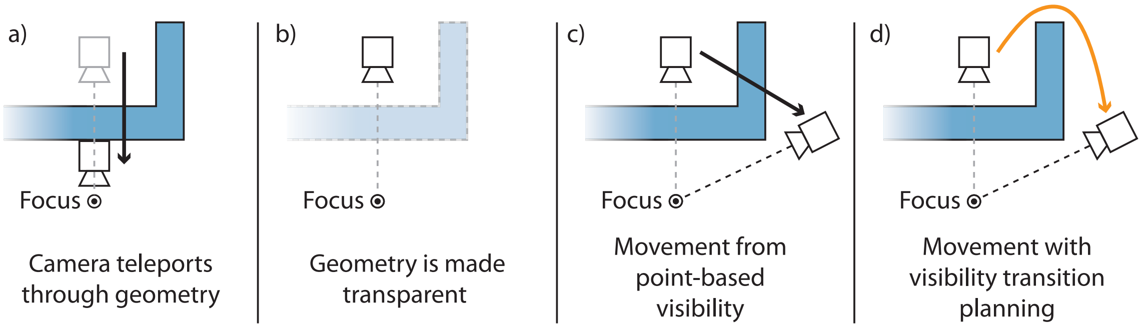

a)
Focus
Camera teleports through geometry
b)
Focus
Geometry is made transparent
c)
Focus
Movement from point-based visibility
d)
Focus
Movement with visibility transition planning

Figure 1: *This simple example illustrates the different behaviors of commonly used methods for camera control. a) The ray-cast can result in sudden jumps. b) The obstructing geometry can be made transparent. c) With point-based visibility [Halper, 2001], the camera may pass through geometry. d) Our method will avoid obstacles correctly.*

problem means that a potentially huge number of visibility evaluations must be considered. Neither the start point, nor the end point, nor the focus point are known in advance, making pre-computation a non-trivial task. Finally, the algorithm has strict, real-time requirements, with only milliseconds available for all computations.

Our approach to control the camera interactively is to apply global knowledge of the scene. In a first part, we define a basic camera planning based on pre-computing a graph with spheres and portals covering the ambient space of the scene. Additional visibility information is stored in the graph. At run-time we perform an extended A^* search [Dechter and Pearl, 1985] that incorporates this visibility information to create smooth, visibility-aware camera transitions. Then, in a second part, we show the extensions we made to our previous work and demonstrate new applications of the dynamic camera control useful in multiple real-time scenarios.

In terms of impact, our contributions include the first camera control system that can generate large, collision-free and dynamic occlusion-aware camera transitions customized for the visibility of a focus point in real-time. This functionality opens up new opportunities for game designers, such as dynamic target switching between multiple characters and fly-throughs that demonstrate a suggested path through an environment from arbitrary start and end points. Additionally, we present a third-person camera routine that optimizes for visibility yet never passes through scene geometry. Existing local camera models cannot make such claims, since some geometric configurations can only be resolved using global knowledge.

Technically, we build on the visibility-aware data structure and modified A^*

search strategy developed in our master thesis [Oskam, 2008]. Our contributions contain a refined path post-processing that yields smoother results and a more in-depth performance analysis of the control algorithm. Furthermore, we contribute advanced search strategies to support moving occluders, and show how the roadmap can be utilized to achieve proactive camera movement that actively seeks a position to prevent the player from escaping visibility. These extensions allow a variety of novel applications for automated camera control.

The remainder of this chapter is organized as follows. First, we discuss related works in camera navigation before we briefly revisit path planning in graphs. Then, we revisiting the visibility transition planning approach by Oskam [2008]. Next, we propose several extensions and applications to the basic algorithm to achieve more advanced and robust camera control. Finally, we show a variety of novel applications before closing with a summary and discussion of the limitations.

1 Camera Navigation Background

In this section we discuss previous work of camera navigation for real-time control, cinematography, visibility-awareness, and motion planning.

Real-time camera control: A thorough overview of the state-of-the-art in camera control for computer graphics is presented by Christie et al. [2009], and we review only the most relevant of these works here. Many classical third-person camera models examine the local vicinity in order to resolve occlusions. A common camera model used in computer games casts a ray from the player to the camera and teleports the camera to closest intersection in order to maintain visibility, leading to a noticeable jump in view. An alternate scheme makes occluding geometry transparent, which avoids camera jumps but detracts from the environment's realism. Halper et al. [2001] present a more sophisticated local camera model that resolves occlusions using point-based visibility. Li and Cheng [2008] focus on finding an unoccluded camera position, but sometimes teleport when the camera cannot be readily moved to this vantage. Marchand and Courty [2002] develop an image-based camera controller that uses visual servoing to resolve visibility tasks and occlusion constraints. These algorithms handle some situations well, but their local nature leads to inherent limitations. The camera may not adequately follow a fast moving character or resolve a complicated visibility situation, resulting in discontinuous jumps or passing through scene geometry (see Figure 1). In contrast, our algorithm is global in nature which permits both large-scale camera transitions as well as a third-person camera that follows an avatar in a natural fashion without discontinuous jumps or collisions with objects in the scene.

Cinematography: More high-level approaches to camera control focus on virtual cinematography, in which cinematographic knowledge is incorporated into the choice of camera position and scene composition. For example, Bares and Lester [1998] present a constraint-based camera planner for shot composition that models different cinematic styles, and He et al. [1996] encode film making heuristics into a hierarchical finite state machine that controls camera selection and placement. Hornung et al. [2003] develop a camera agent for interactive narrative applications. Tomlinson et al. [2000] develop a behavior-based cinematography system that controls both camera and lighting in order to convey a sense of emotion, while Kennedy and Mercer [2001] focus on conveying theme and mood. Our work complements these approaches by offering a sophisticated system for camera transitions. Cinematographic rules may provide a sequence of shot placements and focus targets, while our system controls the actual camera motion between such shots. Drucker and Zeltzer [1994] present an alternative for advanced camera control with visibility based path optimization. However, their approach focuses on the creation of off-line animations and is not suitable for real-time applications.

Visibility: A key aspect of our camera system is the focus on visibility during large camera transitions. Visibility problems are fundamental to computer graphics. As shown by surveys on the topic [Cohen-Or et al., 2003; Bittner, 2002], many visibility algorithms strive to identify which objects, lights, or other portions of a scene are visible from a given vantage. Run-time visibility calculations can be accelerated via pre-computation, in which a visibility relationship between *viewcells* in space and scene objects is established (cf. [Laine, 2005]). Compression techniques for precomputed visibility tables provides efficient storage [van de Panne and Stewart, 1999]. Our algorithm relies heavily on pre-computation for real-time performance. In our setting, however, the focus target is not an object within the scene but rather a point that can be placed anywhere within the ambient space. Our visibility-aware roadmap (Section 2.1) extends the idea of precomputed visibility with a data structure that estimates the visibility of every region of space with respect to every other region of space.

Motion planning: The problem of visibility transition planning is related to the topic of motion planning in the robotics literature where optimal paths are found for robot navigation [LaValle, 2006; Masehian and Sedighizadeh, 2007]. Many motion planning algorithms employ a roadmap construction in which the free configuration space of the robot is mapped to a graph data structure, reducing the planning problem to a graph search [LaValle, 2006]. Some algorithms incorporate a notion of visibility with a sparse set of guard nodes whose visibility region can be defined by unobstructed straight lines in configuration space [Varadhan and Manocha, 2005] or the ability of a local planner to navigate without intersection [Siméon et al., 2000]. The related problem of target tracking strives to compute the motion of a robot observer in order to maintain visibility of a moving target. Sophisticated algorithms address this problem in planar environments with polygonal obstacles (e.g. [Murrieta-Cid et al., 2004; Bandyopadhyay et al., ; 2006]). However, direct extension of this work to full 3D motion is non-trivial, partly because visibility relationships are significantly more complex [Bandyopadhyay et al., 2007]. Other work on 3D tracking does not deal with occlusions [Vidal et al., 2002], utilizes only a robot's visual sensors rather than global scene information [Bandyopadhyay et al., 2007], or presents interesting theoretical results without demonstrating a system that matches the strict efficiency demands of games [Murrieta-Cid et al., 2007; Lazebnik, 2001]. More generally, camera control for real-time graphics has a different set of requirements than robot navigation. A camera control system must operate in complex, 3D environments. Visibility is critically important, and computed paths should always take visibility into consideration. The start, end, and focus point may move continuously. The system must be extremely efficient, with only milliseconds to compute a camera transition and react to scene updates. While the robotics literature has explored navigation in great detail, exiting work does not meet the requirements of a high-performance game camera.

The work of Bandyopadhyay et al. [2007] on 3D target tracking presents an online algorithm designed for an unknown environment where input comes from a robot's visual sensors. In virtual environments, the entire scene is usually known a priori and the camera should make use of this information to find more natural transitions.

Our work is motivated by the navigation system of Salomon et al. [2003]. In their approach, a roadmap is created in which nodes represent an avatar's position, and edges connect nodes between which a local planner can successfully navigate. Niederberger et al. [2004] present a navigation system using a shortest-path search on a triangulated height-field terrain. We build upon these ideas in several ways in order to develop an algorithm that is appropriate for camera control. First, we focus on the full ambient space, rather than just walkable surfaces, and incorporate the notion of dense visibility into the roadmap computation. Rather than a minimal number of guard node visibility points, we favor a dense visibility roadmap in which each node corresponds to a local volume of space that overlaps with the volume of adjacent nodes. Movement within and between these volume bounds is guaranteed to be collision free, and an estimate of the percentage of visibility of all other nodes within a falloff distance is always known. Our runtime planning algorithm uses the precomputed visibility values to find a coarse, global path, and a refinement method makes fine-scale adjustments to improve the path shape and incorporate sub-sphere visibility information. Then, with additional extensions to handle dynamic occluders or incorporating risk prediction for the occlusion of a player, we enable a variety of applications.

2 Camera Motion Planning

Given a start point and end point in the scene. We want the camera to move between those two points while keeping a focus point visible as long as possible. This problem can be solved by employing path planning on a global data structure. In order to move a camera intelligently around a complex environment we build on to of our previous work on visibility transition planning [Oskam, 2008]. First, a graph with spheres and portals covering the ambient space of the scene is pre-computed. In addition, visibility information is stored in the graph structure in order to move the camera such that a focus point is visible as long as possible. Second, at run-time, an extended A^* [Dechter and Pearl, 1985] search is performed that incorporates this visibility information. This initial search provides a coarse path that is then refined to get smooth camera transitions.

2.1 Visibility-Aware Roadmap

The ultimate goal of navigating the camera through a virtual scene is to generate collision-free camera transitions, guided by the visibility of a focus point. Since the start, end, and focus points are specified only at runtime, they can be located anywhere in the scene, and may even change continuously. The planning algorithm may explore arbitrary parts of the environment's free space while continually querying the visibility of the focus point in order to compute the best camera transition. To meet the strict real-time constraints of high-performance applications such as games, a data structure is required that makes these run-time queries as fast as possible.

Motivated by these requirements, visibility transition planning employs a data-structure based on spheres and portals, which is referred to as a visibility-aware roadmap (Figure 2). The entire free space of an environment is tessellated with overlapping spheres. A visibility probability value is computed between every pair of spheres and stored within the data structure. Portals are defined by the circle of overlap between any two spheres. The roadmap is a graph structure derived from the spheres and portals by placing a node at the center of each portal and connecting this node to all other portal nodes associated with either of the two overlapping spheres. By traveling entirely within the network of spheres (transitioning from sphere to sphere via the portals), the camera is guaranteed never to collide with scene geometry. As soon as a focus point is fixed, an estimate of the visibility of the focus point from any query point within the scene is known immediately by looking up

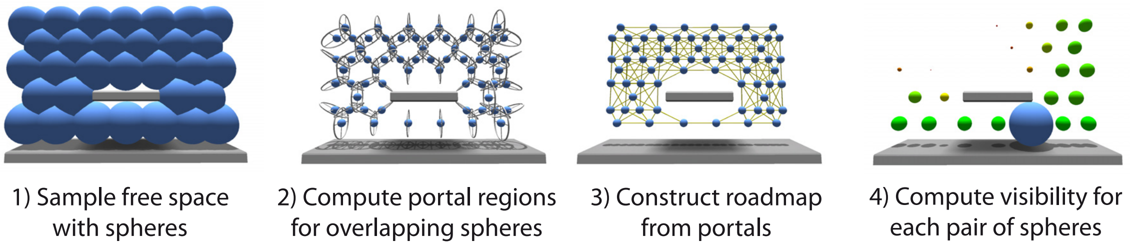

1) Sample free space with spheres
2) Compute portal regions for overlapping spheres
3) Construct roadmap from portals
4) Compute visibility for each pair of spheres

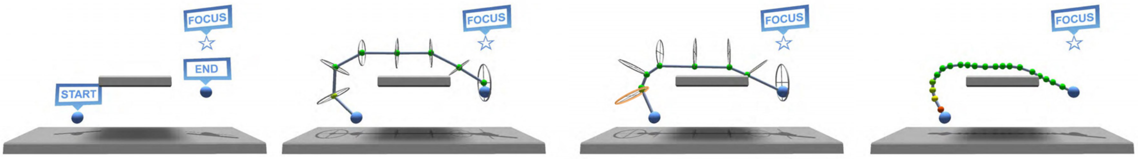
Figure 2: *Overview of the creation algorithm for the visibility-aware roadmap. Based on an initial geometry of the environment, we first compute a spatial discretization. A graph is then built from the overlap regions. Finally, for each pair of spheres, a visibility probability is computed with a Monte-Carlo raytracing algorithm.*

the precomputed visibility probability between the query point's sphere and the focus point's sphere.

Road Map Construction

Unlike existing sphere tree representations [Bradshaw and O'Sullivan, 2004], the visibility-aware roadmap approximates the ambient space with a flat hierarchy of overlapping spheres using an iterative sphere placement algorithm. Although most motion planning algorithms use randomized sampling [Yang and LaValle, 2002], we found that deterministic sphere placement leads to more predictable results, making it easier for game designers to select appropriate parameter values.

First, the scene geometry is embedded within a three-dimensional grid with spacing Δx. Any grid cells that intersect scene geometry are marked as occupied. A candidate sphere of maximal size is constructed at each unoccupied grid cell. To favor uniformity in sphere and portal size, radii are restricted to reflect the levels of detail in the scene, with a minimum radius small enough to resolve fine details in the scene, and a maximum radius that is not larger than the average size of dominant features (e.g. houses or vehicles). A seed sphere is selected at random. Then, in each step of the iteration, the algorithm selects from the candidates the sphere that maximally overlaps the previously selected spheres, creating the largest portals. Grid cells whose center lie within the new sphere are marked as occupied and the corresponding candidates are deleted. The process repeats until no candidate spheres remain. The size of Δx depends on the size of features (e.g. doorways, tunnels, etc.) within the environment and should be chosen to be small enough so that all desired features are resolved by the grid. Although not strictly hierarchical, a finer grid (smaller Δx) can first be used in areas with smaller features (e.g.



1) Compute visibility aware path based on roadmap

2) Construct initial path along portals

3) Refine path to tightly pass through portals

4) Perform final smoothing

Figure 3: *The different steps to compute a visibility aware path based on the roadmap.*

the inside of a house) followed by a coarser grid on the surrounding scene (e.g. the streets of a village).

Adding Visibility Information

A final pre-computation step estimates a visibility probability between all pairs of spheres using a Monte Carlo approach that selects a random point on the hemisphere of a source sphere i facing a destination sphere j. A ray is shot toward a second random point on the opposing hemisphere of sphere j. The visibility probability $p_{i,j}$ between spheres i and j is given by the fraction of rays that reach the destination sphere before hitting an obstacle. To limit the amount of computations for very large environments, we take into account a maximal visibility distance that specifies how much of a level are typically in view. We only pre-compute the visibilities for spheres that are not further away from each other than the maximal visibility distance.

2.2 Camera Transition Planning

Visibility transition planning refers to the problem of finding the shortest collision-free camera transition from a start position s_p to an end position e_p such that a focus point f_p is visible as long as possible. Although the complex nature of visibility in arbitrary 3D environments [Lazebnik, 2001] makes a provably optimal solution to this problem intractable in real-time, we present an algorithm to compute an approximate solution within the strict time constraints of real-time applications. First, our runtime system executes a visibility-based path-planning algorithm on the precomputed roadmap data structure to find a coarse collision-free path through the scene. Next, a fine-scale refinement is performed by computing a sequence of GPU-assisted occlusion maps in spheres of partial visibility. A final smoothing step shortens the path length by allowing it to hug the sphere portals tightly. An overview of these steps is shown in Figure 3.

Path Planning on the Roadmap

The first stage of the runtime system computes a coarse path from the sphere containing s_p to the sphere containing e_p along the visibility-aware roadmap. Due to its efficiency, the A^* search is used to find shortest paths efficiently [Dechter and Pearl, 1985]. The typical shortest-path A^* search uses edge length as the cost function and Euclidean distance to the end node as the heuristic. We augment the edge length cost with the precomputed visibility probability in order to find paths that maximize the visibility of a focus point. The cost for edge e_{ij} between nodes n_i and n_j is given by:

$$C(e_{i,j}) = c(e_{i,j}) + \alpha c \left(1 - u(e_{i,j})\right), \tag{1}$$

where c is the length of the edge $e_{i,j}$ (the Euclidean distance between nodes i and j) and $u(e_{i,j})$ is the visibility probability with respect to the focus point. Due to the construction of the roadmap, each edge lies entirely within a given sphere. Thus, we use $u = p_{S_e,S_f}$, where p_{S_e,S_f} is the precomputed visibility probability between the edge's sphere S_e and the sphere S_f containing the focus point f_p. This value represents the probability of seeing f_p while traveling along $e_{i,j}$. The parameter α determines the relative cost of traveling in regions where f_p is visible versus regions where it is occluded. If α is chosen to be larger than the maximal distance of any path through the roadmap, the algorithm will find the path that travels as quickly as possible into the visibility region. For the heuristic function h of the A^* search, we use the Euclidean distance between the last point on the path and the target:

$$h(n) = ||e_p - x||. \tag{2}$$

Path Refinement

The path planning algorithm yields a path $\mathcal{P}$ along the edges of the roadmap, through the roadmap's spheres. Spheres with a visibility probability of either 0 or 1 are entirely outside or entirely inside of the visibility region with respect to the focus point, while those with a probability between 0 and 1 are in partial visibility. The focus point may be visible from some positions within a sphere of partial visibility and hidden from other positions. Therefore, a detailed refinement step in such spheres is performed so that the computed path navigates along positions where the focus point is actually visible. Since the path planning edge weight favors visibility, there will be, whenever possible,

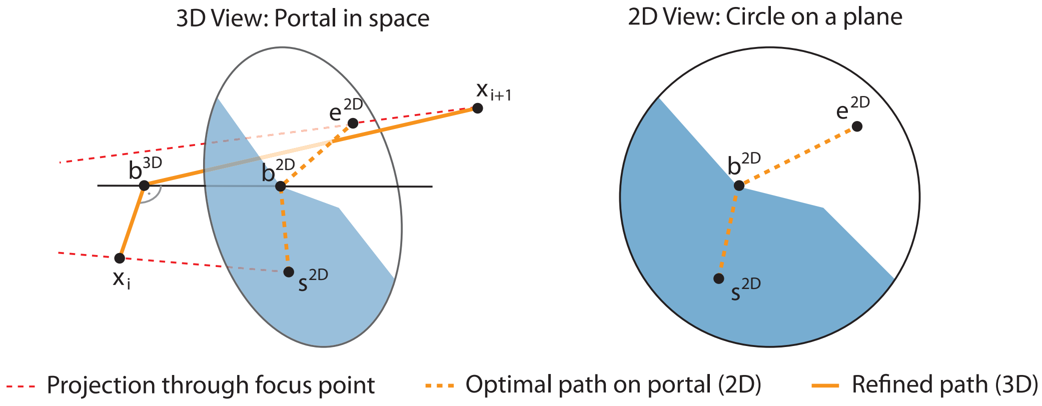


Figure 4: *The distance traveled in the occluded region (shown in blue) is minimized on occlusion maps. The right side shows the 2D plane of the occlusion map with the projections of the path node positions x_i and x_{i+1} (projected from the focus point resulting in s^{2D} and e^{2D}, while the left hand side shows how the 3D positions of the points are computed for the actual path. The path is refined by the point b^{3D} that is a back projection of the point b^{2D} on the occlusion map that minimizes the distance of the path in the occluded part.*

few spheres of partial visibility. Thus, the path refinement need only be performed for a small number of spheres.

The path refinement for spheres with partial visibility can be simplified from a three- to a two-dimensional problem, since one dimension is determined by the line of sight to the focus point. The system builds a detailed representation of the focus point's visibility within the sphere in the form of a 2D occlusion map, which contains per-pixel information indicating whether f_p is visible from a given position within the sphere. The occlusion map is rendered at runtime using a view frustum that is tightly fit around the sphere and originates at the focus point [Halper et al., 2001]. The system performs another A^* search on this occlusion map. 2D path positions on this map that change visibility are detected and reconstructed in 3D.

Although the occlusion maps provide detailed visibility information, rendering them at runtime for every sphere during path planning would be prohibitively expensive because the A^* algorithm may explore hundreds of nodes as it searches for the optimal path. Thus, our system uses the precomputed visibility probability estimates which require only a table lookup during the coarse path planning. Once the coarse path is fixed, only a few spheres will lie in partial visibility due to the nature of the search. The algorithm can afford to compute the more accurate occlusion maps on these few spheres without exceeding the camera's allotted computation budget. Thus, the computation is spent where it is needed most to build the best path.

The start and end points of the 2D path search on the occlusion map are given by projecting the points on $\mathcal{P}$ that enter and exit the sphere onto the map-plane. The entry position x_i and exit position x_{i+1} lie on the two overlap circles of the sphere and its predecessor and successor sphere, respectively. The projected positions are denoted by s^{2D} for the start point on the occlusion map and e^{2D} for the end point in Figure 4. A path planning, similar to the one described in the previous section is performed on the occlusion map pixels, where each pixel is considered connected to its eight neighbors. The distance c and visibility values for u are replaced by functions computed on the occlusion map: c is the 2D Euclidean distance, and u is the average of the two per-pixel visibilities.

Once the occlusion map path has been calculated, the 2D path positions can be reconstructed in 3D. For each pixel, the 3D position can lie anywhere on its projection ray toward the focus point within the sphere. The reconstruction of the start and end points, s^{2D} and e^{2D}, are known from the projections of their 3D positions x_i and x_{i+1} onto the map.

Next, border points b_i^{2D} are identified on the 2D path. Border points are points on the path where the visibility changes from occluded to visible, or vice versa. Depending on the landscape of occluded regions on the projection map, there can occur one or two such border points. One border point appears when the 2D start point s^{2D} is occluded and the 2D end point e^{2D} is visible, or vice versa. Two border points appear when both s^{2D} and e^{2D} are occluded, but a region on the occlusion map is visible (or both points are visible and there is an occluded region separating them on the map). In any case, each occluded region on the map, such a border point is minimizing the path length in the occluded area. This implies that the occluded part is a straight line. That means that for the construction of the $3D$ position of the border point and the path segment in the occlusion region it is enough to project only the border poins to 3D. The 3D positions b_i^{3D} are given by the closest point on their view-line segment (connecting b_i^{2D} with the focus position f_p) to either x_i, or x_{i+1}, as shown in Figure 4.

On the other hand, the portions of the 2D path that are fully visible do not necessarily form a straight line. To avoid errors introduced by approximating visible portions of the path also by a line between b^{2D} and its neighbor, additional points can be iteratively inserted in 2D and reconstructed in 3D as the closest point to the line formed by its 3D predecessor and successor.

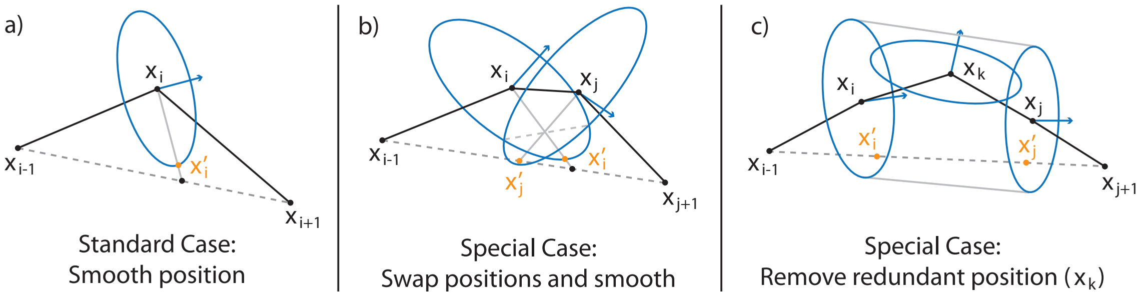


Figure 5: *The path post-processing computes smoothed point positions x'_i as the closest point on the portal to the direct connection of the previous position x_{i-1} and next position x_{i+1} in the path. a) The standard case of smoothing on a single portal. b) The procedure for two overlapping portal circles, where the order of x_i and x_j is swapped during smoothing. c) The procedure for a circle that lies within the tube of the two adjacent circles. This portal does not contribute to the path and, therefore, is removed. The remaining points are then smoothed according to the cases a) or b).*

2.3 Path Post-processing

Although the coarse planning and refinement determines the gross nature of the camera path, the actual path traversed by the camera can be freely moved anywhere within the selected portals without colliding with geometry or changing the visibility score. These additional degrees of freedom can be used to smooth the path, creating both shorter and more natural camera movement.

Path Shrinking

The path positions x_i are computed using a constrained iterative smoothing algorithm. The corrected position x'_i of each point x_i is first found as the intersection of the line from x_{i-1} to x_{i+1} with the corresponding portal's plane. If the intersection point lies outside of the portal circle, it is moved to the nearest point on the circle boundary, as shown in Figure 5 a. Note that due to the previous refinement of the path in partially visible spheres, either of x_i's neighbors can be a reconstructed border point. These steps are performed iteratively for all points of the path. This update can change the projected start and end positions on the occlusion maps, so the path refinement as described in Section 2.2 is re-computed in an additional pass after each iteration.

Two special cases of the post-processing are distinguished. The first one is that of two overlapping portal circles. In such a situation, two neighboring points on $\mathcal{P}$, x_i and x_j, can converge to a single point on the intersection line of the two circles, which prevents the points from moving further toward their smoothed positions. To resolve this problem, a combined update of both points can be performed if they are closer than a small distance ϵ (Figure 5 b). The second special case occurs when a portal is completely contained in the cylindrical volume of its two neighboring portal circles. As the contained portal does not contribute to the overall volume of the path, we simply discard it (Figure 5 c).

Final Smoothing

The final camera trajectory is determined by Hermite interpolation [Farin, 1990] of the path between each pair of points. Two consecutive portals cannot lie on the same plane, which guarantees that a C_1 continuous interpolation can be found. The maximum curvature is bounded by the space between the two portals. In partially visible spheres, where the projection map forces the border point onto the margin of the sphere, there might not be enough room to add additional control points for the Hermite interpolation. In this case, in order to guarantee the C_1 continuity of the final path, the border point can be slightly moved toward the sphere center to create enough room for the interpolation. Figure 6 shows a visualization of this final smoothing.

In order to interpolate between the two points x_i and x_j in $\mathcal{P}$, the tangents m_i and m_j of the path at these positions need to be estimated. We perform finite differences between the points neighbors in $\mathcal{P}$:

$$m_i = ||x_{i+1} - x_{i-1}||. \tag{3}$$

In the case of the start point s_p or the end point of the path, where only one of the two neighbors exists, the tangent can be computed as finite difference between the start or end position and the only neighbor point. Now, given all the tangents m_i, the path can be refined piecewise between two points by inserting new points. In our implementation we insert equidistant points. Given the distance $d(x_i, x_j)$ between the two nodes, we can compute a factor k between $[0;1]$ for each point to be inserted, where $k = 0$ corresponds to x_i and $k = 1$ to x_j. The new positions $x_{i,j}^k$ between x_i and x_j can now be computed as

$$x_{i,j}^k = (1-k)^3 b_0 + 3k(1-k)^2 b_1 + 3k^2(1-k)b2 + k^3 b_3, \tag{4}$$

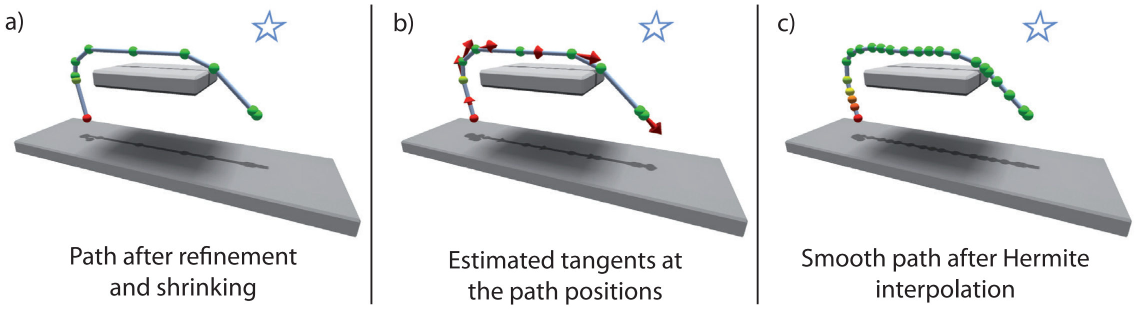

a)
Path after refinement and shrinking
b)
Estimated tangents at the path positions
c)
Smooth path after Hermite interpolation

Figure 6: *Visualization of the final smoothing step on an example path. a) The path positions resulting from the path refinement and shrinking are sparse and can cause kinks in the path. b) First, the tangents at the path positions are estimated using finite differences. c) Second, Hermite interpolation is applied to yield a C_1 continuous path that still passes through the original path positions.*

where the points b_0, b_1, b_2, and b_3 are the bezier points. They can be computed as follows

$$\begin{aligned} b_0 &= x_i \\ b_1 &= x_i + \frac{d(x_i, x_j) m_i}{2} \\ b_2 &= x_j + \frac{d(x_i, x_j) m_j}{2} \\ b_3 &= x_j. \end{aligned}$$

This yields a smooth, C_1 continuous path that still passes through the original path positions x_i. To also propagate the visibility values v_i and v_j from the position pairs (x_i, x_j) to the new inserted positions $x^k_{i,j}$, we employ linear interpolation using the Euclidean distance between the nodes.

$$v^k_{i,j} = \frac{||x_i - x^k_{i,j}||v_i + ||x_j - x^k_{i,j}||v_j}{||x_i - x^k_{i,j}|| + ||x_j - x^k_{i,j}||} \tag{5}$$

3 Extended Planning Strategies

In the previous section, we have discussed the basic framework for visibility transition planning, which includes the precomputed visibility roadmap as well as a runtime planning, refinement, and smoothing strategy for visibility-aware camera transitions. This functionality is useful in many situations as-is. However, interactive environments, simulations, and computer games often have unique, specialized requirements, and one single camera model cannot satisfy all situations.

Our data structure and basic planning system provide the foundation and algorithmic tools to enable a variety of customized camera behavior that can be specialized to the needs of a particular application. In this section, we explore several extensions to our basic planning framework that allow a variety of applications in dynamic environments.

3.1 Dynamic Planning

The visibility-aware roadmap and the transition planning provides a tool to compute paths for a given start, end, and focus point in the scene. This is useful, for example, for navigating in urban environments such as google earth as demonstrated in [Oskam, 2008]. The visibility planning can create camera paths from a persons location to his or her favorite restaurant in a global context, allowing the user to see the restaurant's location as early as possible. The inverse case, where the user wants to see his location in greater context, can be computed using visibility transition planning as well. In this case the user's position is kept in shot as long as possible. Such transitions could assist in way finding tasks in virtual environments by providing a natural facility to connect egocentric and allocentric viewpoints without losing context [Byrne and Becker, 2008].

In these examples, the camera path is traversed from beginning to end once it is computed. However, if either of the start, end, or focus points moves during the traversal, the path is not valid anymore. To be able to react to such changes instantly, the path needs to be updated while traveling. This is achieved by having the planning constantly active in a thread running parallel to the program's main loop. This separation decouples the complexity of the visibility planning from the application's frame rate constraints. A straight forward threading of the search, however, is not possible because of the rendering of the occlusion maps. Their rendering requires synchronization of the GPU usage with the main rendering step, as also the main thread utilizes the graphics card for rendering the scene.

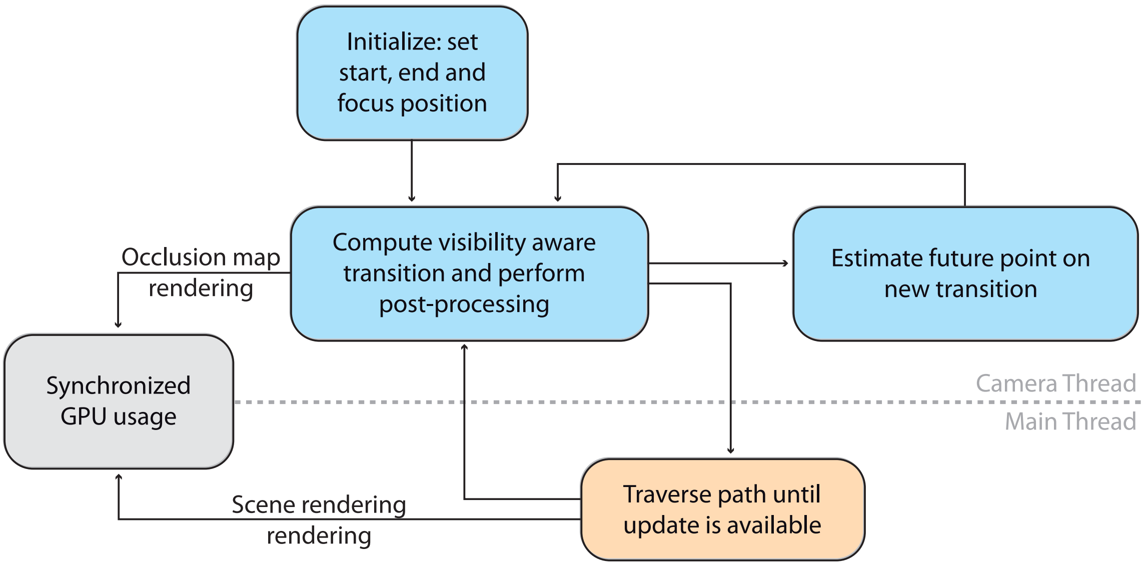

Initialize: set start, end and focus position
Compute visibility aware transition and perform post-processing
Estimate future point on new transition
Occlusion map rendering
Synchronized GPU usage
Camera Thread
Main Thread
Traverse path until update is available
Scene rendering rendering

Figure 7: *Implementation of the dynamic visibility transition planning. By predicting a future start point on the current path based on the time it took to compute the path, a start position for the next search can be estimated. The new transition path is computed in parallel while the camera travels along the current path. The computation of the occlusion maps, which is performed using the GPU, needs synchronization with the rendering of the main thread.*

Figure 7 shows the state machine that implements proper synchronization between main thread and planning thread. First, the state machine is initialized with the initial start, end, and focus positions. These are used to compute an initial path. Based on the time t it takes to compute this first path and a given maximum velocity v of the camera, a future point d on the initial path is estimated.

$$d = t\,v\,\alpha \tag{6}$$

We compute the distance d on the path using a security buffer α. This buffer assures that fluctuations in the time the new path is computed do not affect the global smoothness of the camera movement. These fluctuations may be caused by waiting times by the state machine for synchronization of the GPU usage or environmental factors such as the focus point vanishing behind an occluder. In our implementation, a value of $\alpha = 1.5$ sufficed to allow a smooth camera motion while maintaining dynamic adaptation.

This new point d on the path currently travelled by the camera can now be used as start point for the new search, while the end and focus point are updated according to their movement in the scene. As soon as the new path is ready, it is given to the main thread so that the camera can switch the transition as soon as d is reached on the old path. At this point, the new point d is estimated and the state machine restarts. For the estimation of the new d, the distance the camera was not able to travel towards the old d can be taken into account as well, to not cumulate buffer distances over the course of a long path.

Since both the camera and the main threads require GPU time, they have to be synchronized at key points. The main thread, since it is crucial that it runs with a smooth frame rate, is used as pacemaker for the camera thread. Each time the camera thread needs occlusion maps, it is put on hold, until the game engine thread cycles through the next draw call. Then, all the pending occlusion map renderings are called from the main thread before the scene is drawn.

The dynamic planning enables a new level of usability of the camera planning. We have designed the dynamic state machine to be independent of f the planning algorithm. The interface consists of the standard search input, the start, end, and focus points. Also the required synchronization of the GPU usage can be extended with other tasks by simply adding the requests to the list of occlusion map rendering. This design allows arbitrary extensions of the search strategy to work with this dynamic state machine. The changes only need to be formulated within the original framework. The extensions discussed in the following are all compatible with the dynamic planning, and, in Section 4, we show a series of applications and results that are made possible through their combination.

3.2 Avoiding Occluders

Often, real-time environments contain dynamic elements, such as closing doors or moving obstacles, and it is crucial that the camera takes these objects into account when moving through the scene. Fully dynamic environments where every environmental feature can move are uncommon since they invalidate acceleration structures necessary for collision detection, lighting, and other computations. Thus, we target the most common and advantageous case where the environment contains a small number of dynamic occluders.

To enable the computation of visibility transitions in the presence of dynamic occluders, two problems in the visibility-aware roadmap need to be solved: invalid connections, and invalid visibility. On the one hand, a dynamic object in the scene moving through the ambient space. This space is occupied by spheres in our data structure that cover a volume where our camera

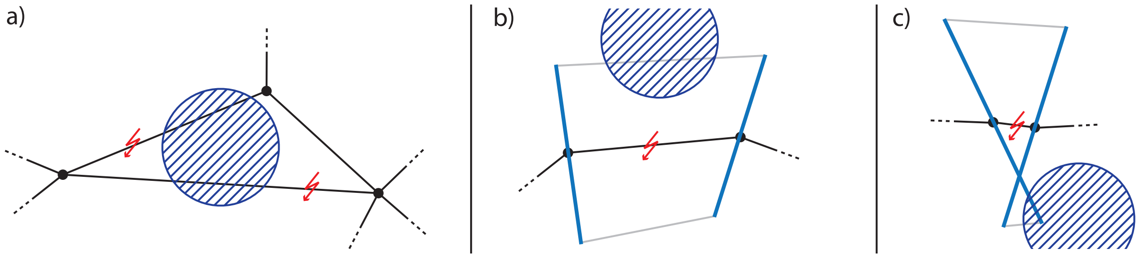


Figure 8: *Three different cases for the removal of roadmap edges are shown. a) Edges that are colliding with occluder are directly removed from the graph. b) Additionally, edges, whose corresponding cylinder (enclosed by the node's portals) collides with an occluder are also removed. c) The special case is shown where the portals are crossing. Since the post-processing may cause the path to converge into the lower portion, this collision also removes the edge from the roadmap.*

can move freely. This property is invalidated by the occluder that collides with connections in the roadmap. On the other hand, the object changes the visibility of the scene. Our visibility-aware roadmap contains visibility probability information between each pair of spheres. The occluder changes this value as it moves through the scene. These two problems can be solved separately.

Invalid Connections

In order to remove connections in the graph, the bounding sphere of the colliding object can be considered. In a first step, all connections in the graph can be invalidated that intersect with the collider (as demonstrated in Figure 8 a). This way, the planning will automatically circumvent the occluder. Additionally, if it can be assumed that the occluder is moving with finite velocity, the collision tests can be performed incrementally from one frame to the next, since the roadmap allows fast querying of the neighborhood.

While this straight forward canceling of roadmap edges leads to a correct result after the initial path planning search, the smoothing of the path afterwards can cause problems. An occluder, that is not intersecting an edge, may still intersect the volume formed by the two portal circles of the edge's two nodes (two cases are shown in Figure 8 b) and c). Such a configuration allows the possibility that the smoothing causes the path to drift into the occluder. Therefore, roadmap connections also need to be removed from the search if the occluder collides with the volume between the two portals. Such an intersection can be found by performing the following test (we assume a spherical occluder):

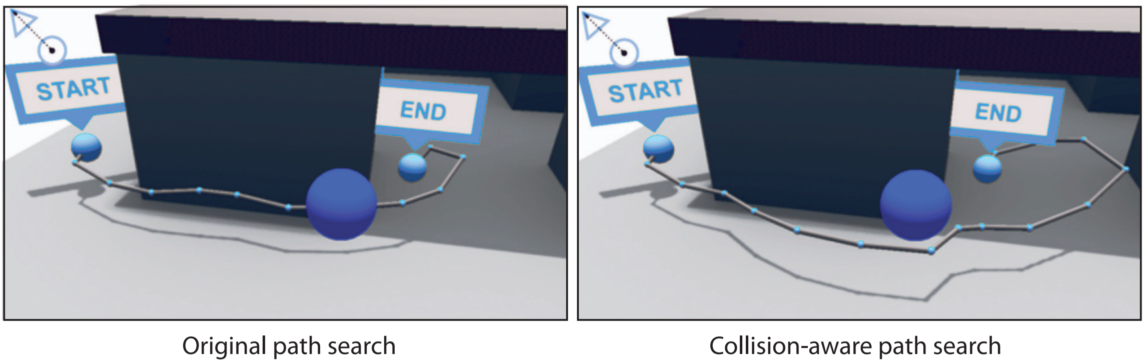


Figure 9: *Comparison of the result from the original path search (left) and the collision-aware search (right).*

1. Test for an intersection of the occluder against the sphere enclosing both portals. If an intersection is detected, continue with 2.
2. Test for an intersection of the occluder against either of the two portal circles by finding the closest points of the occluder to the circles plane and projecting it onto the circles border. If an intersection is detected, invalidate the roadmap edge. Else, continue with 3.
3. Connect the two points on either circle found in 2 and test for an intersection of the line against the occluder. If an intersection is detected, invalidate the roadmap edge. Else, continue with 4.
4. Compute the intersection circles of the occluders with both planes of the portals. If either of the circles center is closer to the node than the corresponding portal's radius, invalidate the roadmap edge. Else, continue with 5.
5. Test if the occluder is completely inside the volume by testing if the distance of the occluders center is closer to the roadmap edge than the line used in 3.

This intersection test, similar to the canceling of roadmap edges directly, can be performed incrementally on the data structure for each frame to further decrease computational complexity. With the edges in danger of causing collisions with an occluder removed from the graph, the path planning can be executed as is. Figure 9 shows a comparison of a search with and without canceling of colliding edges.

Adapted Visibility

At this point the roadmap can be made aware of occluders in the scene that cause both the search and the smoothing to produce a camera transition that collides with occluders approximated by spheres. The camera paths produced at this point, however, can have suboptimal visibility to the focus

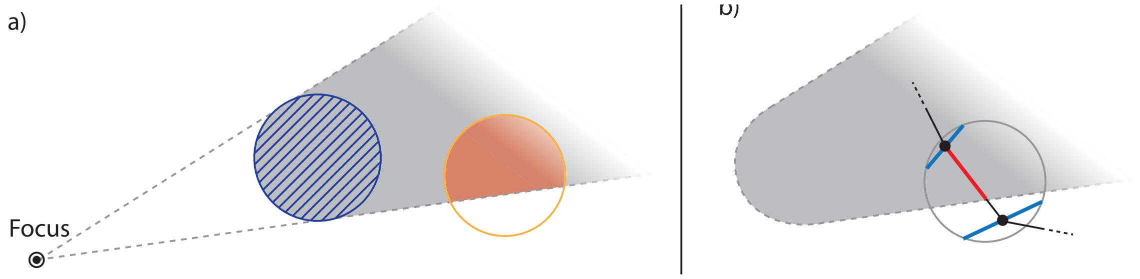


Figure 10: *An occluder creates a conical region in the scene where visibility to the focus point is broken. a) In order to update the visibility probability of the graph's spheres, the overlapping volume of the sphere and cone could be calculated. b) In our framework, we approximate this by calculating the fraction of the edge that lies inside the visibility shadow.*

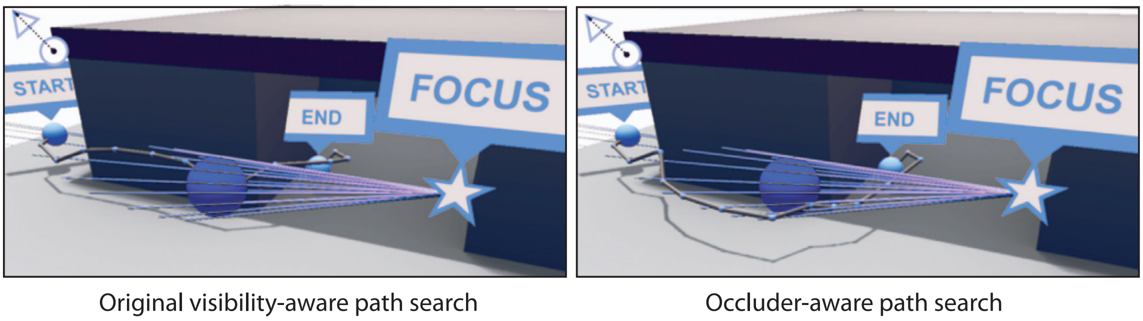

START
END
FOCUS
START
END
FOCUS
Original visibility-aware path search
Occluder-aware path search

Figure 11: *Comparison of the result from the original visibility path search (left) and the occlusion-aware visibility path search (right). The rays originating in the focus point visualize the cone created by the occluder.*

point. The occluder throws a shadow of invisibility into the scene. This shadow needs to be considered during the search, lowering the visibility of the affected edges. The edge visibility u already contains the probability information from the environment. It can only be lowered when affected by the occluder. Therefore, we model the visibility of an edge by multiplying a penalty term p to compute the new edge visibility $\tilde{u}$

$$\tilde{u} = u\, p. \tag{7}$$

The occluder in the scene creates a cone segment in the scene originating in the focus point where the visibility is zero. This is shown in Figure 10 a). The sphere S_e that contains an edge affected by this visibility shadow intersects the cone. The proper value of the penalty p could be computed as the fraction of S_e that intersects the cone segment. However, since the cone has an irregular shape, the exact solution is computationally expensive.

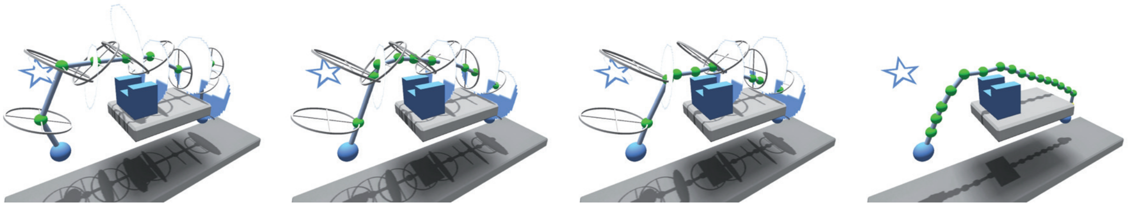


Figure 12: *In this example, the occluder is a U shaped block. From left to right, snap shots of the path shrinking and smoothing are shown. This example shows, that considering the exact geometry of an occluder during smoothing allows the path to adapt to the visibility on a higher resolution than the graph connections alone offer.*

Therefore, we approximate this by computing the fraction of the edge affected by the cone (see Figure 10 b). This solution is easy to compute and leads to plausible results. Figure 11 shows how the path changes when both dynamic collisions and occlusions are taken into account.

Extended Path Smoothing

To this end, only spherical occluders have been considered. However, objects may have more complicated shapes. In these cases, they are usually approximated by several spheres. Nonetheless, the exact shape of an occluder can be taken into account. During smoothing, one crucial aspect is the utilization of occlusion maps seen from the focus point to refine the path. Here, the occluder's exact shape can be taken into account by adding them to the environment geometry for the occlusion rendering. This way, the path refinement can be executed as is, as shown in Figure 12.

Using the extensions to invalidate roadmap edges and include the changes to visibility, we are able to execute the camera planning in the same way as in the basic framework. With the awareness of dynamic occluder, motion planning can be used in more complicated cases, where a goal can be achieved even with certain dynamics in the scene occurring randomly.

3.3 Hierarchical Search Criteria

As we have discussed in the previous sections, the visibility transition planning framework is able to dynamically create camera transitions that optimize for visibility to an object. The concentration on visibility is very useful if seeing the focus point is the prime interest. There are scenarios, however,

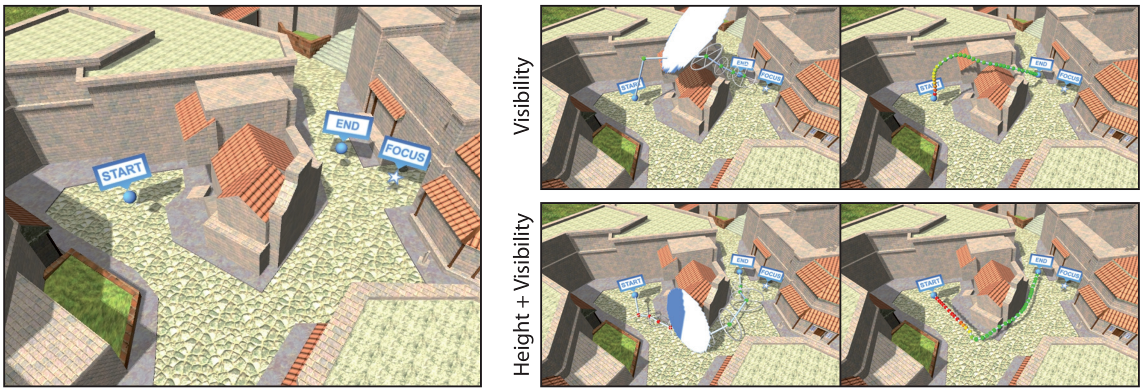


Figure 13: *An example of hierarchical search criteria is shown. The image on the left shows the situation with the focus point near the end point hidden behind a building. While the standard framework provides the quickest path into visibility (top), a hierarchical height constraint allows to find a solution that still seeks visibility, but stays near the ground.*

where prioritizing visibility at all costs can be counter productive. An example is when the camera is moved in a flat but structured environment, like a town. There are buildings and alleys that create a network of passages and provide a clear structure to a visitor traveling on the ground. If the camera is moved to a location of interest, it will always move straight in the air until it sees the focus object before it transitions to the appointed position. This behavior leads to repetition in the camera motion that can become boring and predictive over time. In such situations an application might need additional constraints for the motion planning. A path through the town to an interesting location could require the camera to move along the ground, in a similar way that a player would do, instead of taking the short cut over the roofs.

The search framework can be modified to include such additional constraints. The cost function of the A^* search encodes the way an optimal path is found. By modifying the edge cost appropriately, an optimal path can be controlled using additional criteria. For the example above, where the camera path should lead through alleys, a height constraint could be added that weights the cost of an edge according to its distance from the ground. To achieve such a behavior, we add an additional penalty term to the edge cost in Equation 1. This leads to a modified cost function

$$C'(e_{ij}) = C(e_{ij}) + \alpha^2 c_{ij} h(e_{ij}) \ , \tag{8}$$

where $h(e)$ evaluates to zero if the edge e is within a given height range, and increases to one if the edge is above or below this region. The weight of α^2

ensures that this constraint is prioritized over the visibility. The additional search criteria compared to the original visibility only planning is shown in Figure 13. It can be observed that the visibility to the focus point is still optimized, but within the height constraint.

This hierarchical search of different constraints can easily be extended by a variety of penalty terms, such as a weight term specified by a level designer to make the camera prefer landmarks in the environment seen from a certain angle or region. The exponent of the weight α used in the hierarchical cost function allows to specify the priority of a constraint.

3.4 Risk Prediction

So far, we have concentrated on large camera transitions, as this is still an open problem for dynamic environments. One large branch of interactive applications, namely computer games, however, mostly contain local camera behavior. Usually, the camera is following a user controlled avatar with a predetermined distance. In some cases, the user is allowed to change the camera orientation, and in other cases, the camera's orientation angle with respect to the user is changed automatically. In either case, the camera behavior is constrained to the local vicinity of the player. At first glance, a local solution seems to suffice, where simple heuristics adjust the camera when occlusions occur [Halper et al., 2001]. Figure 1 a) through c) shows some examples of local camera behavior. However, global planning can improve local camera control by guiding the camera in situations where local models can fail (1 d).

Generally, local heuristics use the fact that the player only moves a limited distance in one frame. Should the player be occluded, then the camera only has to adjust slightly to resume a clear view. This assumption is true in most cases. However, the camera is a certain distance away from the avatar, and if the player quickly dashes behind a close object, the camera might need to be forced to move a large distance. This is where local methods can fail, and global information is required to improve the camera behavior.

The dynamic planning discussed in Section 3.1 alone can already be used to improve local camera control. The current camera position at frame t, as determined by the local controller, is defining the start position. The supposed camera position at frame $t + 1$, also determined by the local controller, marks the end position. The focus point is the avatar itself to which the camera is attached. Now, in cases where the start and end points do not lie in the same sphere, visibility planning can be activated.

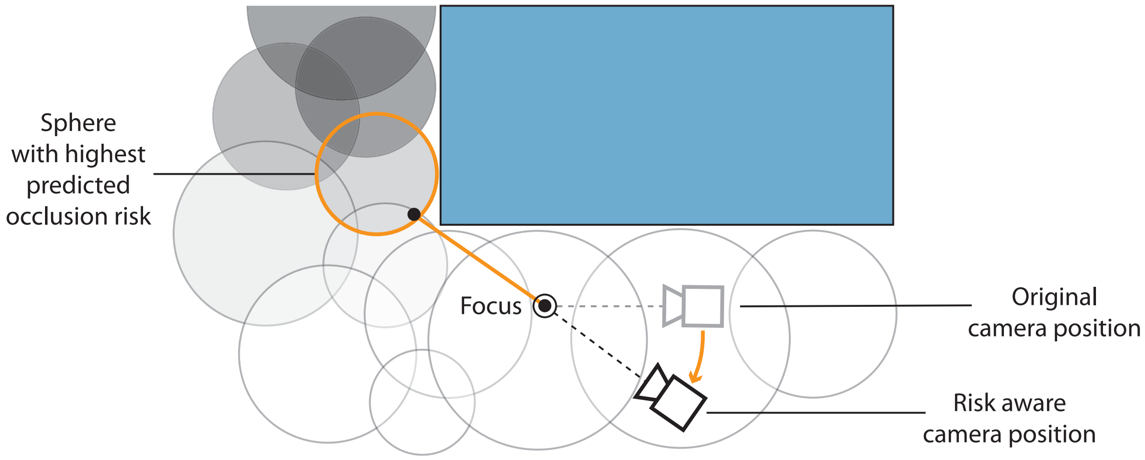


Figure 14: *Proactive camera movement: our algorithm finds the closest sphere (indicated in orange) that has reduced visibility from the original camera position. After detecting such a high escape risk, the camera's viewing direction is aligned towards this sphere. Spheres with reduced visibility from the original camera position are shown in darker shades of gray.*

However, a purely reactive camera controller cannot keep a fast moving player in view. Usually, local camera models purely react to changes of the player's position, but cannot proactively prevent the player from moving into an occluded region. Thus, also the visibility planning, which is activated in such a case, will be purely reactive.

The data structure we compute for visibility aware motion planning allows to detect situations where the avatar is in risk of occlusion. We can approximate the shortest path to an occluded region, similar to approaches in the robotics field [Bandyopadhyay et al., 2006]. This means that we can predict which direction has the highest risk to occlude the player, and move the camera before the player enters the occluded region. This concept is visualized in Figure 14.

Similar to visibility transition planning, we perform a search on the roadmap to compute the path to the closest point in the occluded region. Here, any graph search strategy can be used. Since there is no indication about the end point of the search, we cannot speed up the search with a heuristic. There are, however, several early abort tests to limit the search. One constraint for the camera is that it should be placed behind the player, to show the region where the player is moving towards. Thus, the search can be restricted to the hemisphere given by the current velocity of the player. In addition, only occluding regions close to the player are relevant. So we can limit the path search to compute paths shorter than a given distance. An appropriate distance can easily be estimated by the players maximum velocity.

In our implementation, we have used a maximum distance of $d_{max} = c\, p_v + r_{max}$, where p_v is the current velocity of the player's avatar, and r_{max} is the largest radius of a sphere in the visibility-aware roadmap. For c we have found that a value of 65 units (approx. 10% of the environments width) yields good results for our player model with a maximal speed of approximately 1 unit per frame (the player's bounding sphere being 2 units). The first sphere s, with a visibility probability less than p_p to the sphere containing the camera, that is encountered during the search is selected. This search yields the path with the highest risk to occlude the player. Adjusting p_p allows to adjust the sensitivity of the risk predictions. A value close to 1 will cause the camera to adjust often to corners in the players vicinity, while a value close to 0 will only cause a reaction when the player is close to a corner and traveling with high speed. In our experiments we achieved a good balance with a visibility threshold of $p_p = 0.5$.

As soon as a high-risk sphere is determined, we adjust the camera by manipulating the local camera controller. The direction r towards the occluding region is given by the vector from the player towards the center of the sphere s. This vector could be used directly to compute a desired position for the camera, but typically, the camera is preferred at a fixed angle above the ground (according to the local model that defines the camera behavior in the first place). We thus project the risk vector r onto the ground plane and set the camera position according to the local model, looking along r with the given angle above the ground plane.

In environments where the player is not only moving along the ground dimensions, but is able to navigate in all three dimensions, we propose to adjust the plane used to determine the camera behavior in risk situations. We have implemented a virtual plane that is defined by the players velocity vector and the scene up vector. The risk vector r is then projected onto the virtual plane instead of the ground to determine the direction of the camera.

The risk prediction is applied using a simple adjustment of the camera position. This can help in environments with a few distinct corners. However, it might fail in cases where the player quickly dashes along a corner without moving behind it. Here, the camera adjusts to a false alarm. However, our risk prediction delivers one or more spheres where the player might get occluded. This information could be utilized in a more sophisticated way, like adding an extra risk value to corners that have been travelled a lot by the player, or using a different approach in adjusting the camera once a high-risk sphere is detected.

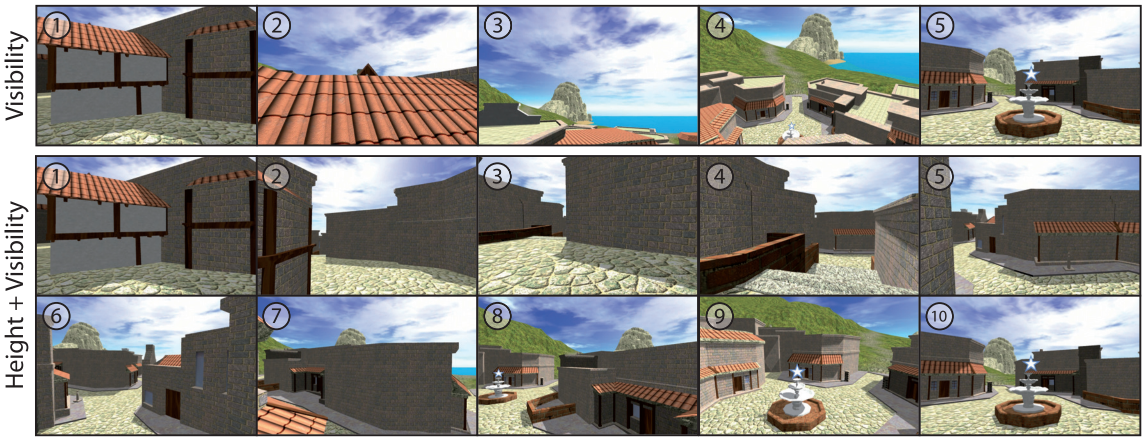


Figure 15: *Comparison between original visibility only planning and visibility planning with a constraint for height. The camera flight on the top takes the shortcut over the roofs while the camera transition on the bottom follows the alley to the focus point.*

4 Applications and Results

The camera planning framework presented in Section 2 and the extended planning strategies in Section 3 allow a wide range of applications. As we have formulated our camera motion control as operations on a specialized global data structure, we are able to automate certain camera behavior that was previously not possible. In this section, we demonstrate applications of our framework to improve local camera behavior, allow constrained global camera transitions in dynamic environments, and increase awareness of the context when spectating interactive content at run-time.

4.1 Content-aware Camera Transitions

Our camera planning framework, per design, is aware of the global context of the scene. The visibility-aware roadmap allows to plan the camera movement through the scene without colliding with geometry. Through the use of this global visibility-aware planning combined with our dynamic update of the path at run-time it becomes possible to automatically generate camera work. Based on the application one or more of the extensions presented in Section 3 can be added to achieve certain results.

An example of a camera path that should show the location of a point of interest in a small town is shown in Figure 15. Here, the player is standing at a corner and would like to know how to get to the town well. Using only

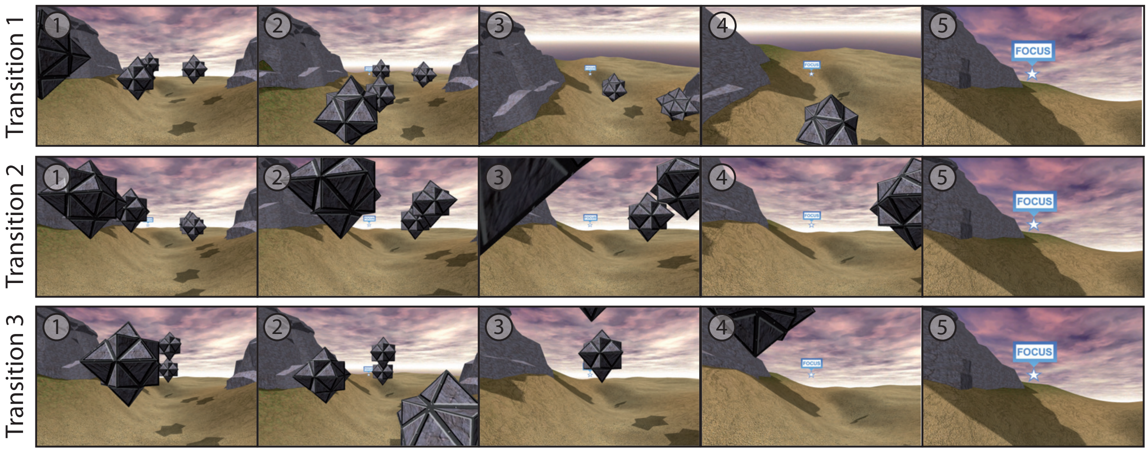


Figure 16: *Dynamic camera motion avoiding randomly moving blocks. The goal is to move the camera towards the focus object and directly transition through the moving blocks. Our dynamic camera motion framework is able to guide the camera through the blocks at different instances in time without occlusions or collisions.*

visibility-aware transition planning, the camera moves over the roofs directly to the focus point. Although the transition is smooth, the player does not get enough information to find out how to get to the well. The same example, but using a constraint for height causes the camera to fly through the alleys until it reaches the well. This transition is still aware of visibility, but prioritizes the height, resulting in a much more informative camera transition for the player.

In many real-time applications the designer or programmer intents for certain events to take place. A common example is a boss fight in a computer game. The event has to take place in order to progress, and, therefore, it is usually important and should be put in frame nicely. The problem that limits the artist from defining camera positions and transitions is, that the environment may not be completely controllable. Doors may be closed or open, objects may be at different places, or even the environment itself may have dynamics that prevent certain shots.

Our dynamic occluder avoidance extension allows the camera to adapt to the unknown situation, and create dynamic moves to enhance the cinematographic quality. An example is shown in Figure 16. The goal is to have the camera move through the randomly flying stone blocks and zoom in on the focus of attention. Started at three different instances in time, our controller is able to guide the camera to both avoid occlusions of the focus point and collisions with the blocks.

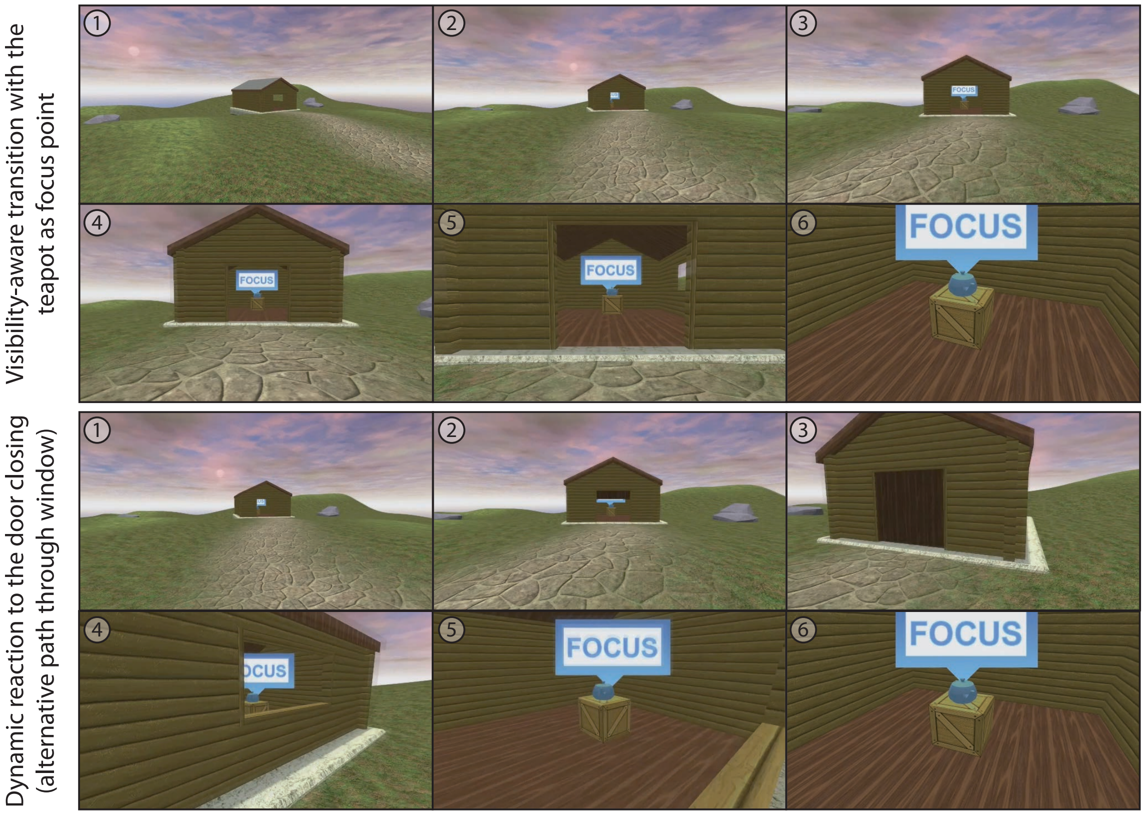

Visibility-aware transition with the teapot as focus point
Dynamic reaction to the door closing (alternative path through window)
1
2
3
4
5
6
FOCUS

Figure 17: *In interactive environments it is sometimes crucial to provide hints on hidden objects. In this example, we demonstrate our camera planning by showing the teapot hidden in the house. If the door closes while the camera transition is executed, our dynamic planning is able to find an alternative route.*

Another example is shown in Figure 17. The goal of this scene is to show the location of the hidden teapot to the user. The designer of the virtual environment only wants the user to become aware of the teapot, but does not want to worry about the exact state of the environment. Using our framework, the designer only needs to define what to focus the camera on and a trigger (e.g. a hit box in the environment) in order to generate the appropriate camera transition. The top example in Figure 17 shows the camera transition that spots the teapot in the house and moves towards the focus point. The images on the bottom show that our framework is able to instantly react to unforeseen events, quickly finding an alternative route to the desired focus point.

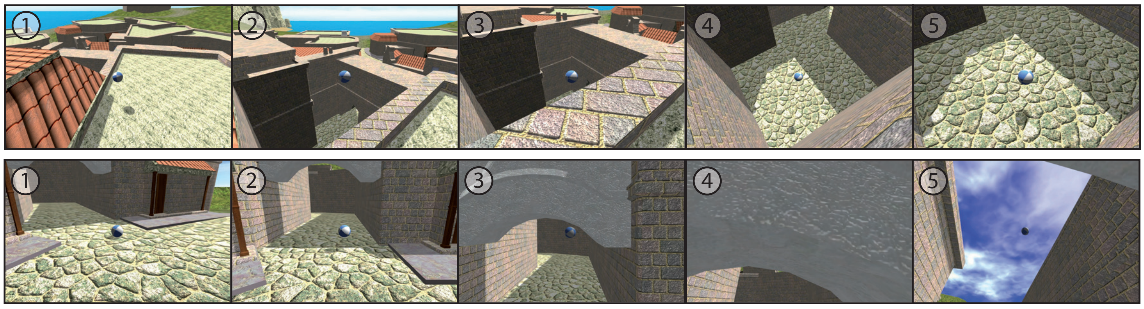


Figure 18: *Two examples of improved local camera control using our dynamic planning framework. The camera is mounted to the avatar using a standard ray-cast camera. In any situation, the camera moves smoothly.*

4.2 Improved Local Camera Behavior

As discussed in Section 3.4, our global planning in combination with the dynamic update can be used to improve local camera models. The local control may be any method that provides a position and direction for the camera. Both parameter do not necessarily need to be continuous. Our planning framework will make sure that the camera always moves smoothly and optimal with respect to visibility in order to connect discontinuities in the local controller. This is demonstrated in Figure 18, where the camera is constrained to the avatar by the local ray-cast camera model. In the ray-cast model, the camera is always at the same distance and angle behind the avatar. If an objects gets in between the camera and the avatar, the camera is automatically teleported in front of the nearest occluder. This is a popular camera model as it is simple to implement and cheap to compute. In the top example of Figure 18, the avatar jumps down close to a ledge. While the local controller causes the camera to be teleported, our global planning smoothly guides the camera behind the avatar. The second example in Figure 18 is similar, as the player dashes up an arch. Our dynamic planning guides the camera smoothly underneath the ledge to catch up with the player behind it.

A purely reactive camera controller cannot keep a fast moving player in view at all times. In order to anticipate situations in which the player might escape the camera's region of visibility, our system uses risk prediction. A series of snapshots from a scene that demonstrate this extension are shown in Figure 19. The camera is fixed behind the player at a certain angle. The player moves quickly forward between two horizontal platforms, and, suddenly, dashes up. Usually, the camera would hit the upper platform, and leave the player occluded for a moment. With the risk prediction, however, the camera is moved beforehand, and an occlusion is avoided. We have found environments that are complex in all three dimensions (contrary to town environments, that

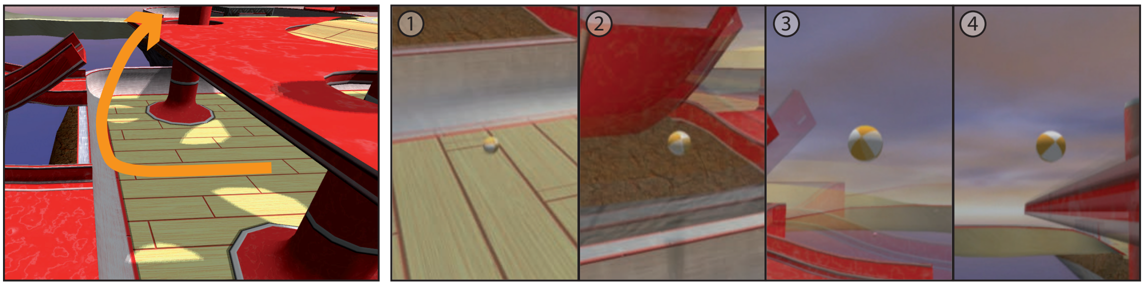


Figure 19: *Example of active risk prediction. The image on the left shows the situation where the player is moving quickly from underneath a platform and turns upwards. The image series on the right shows, that the risk prediction is able to move the camera to avoid getting occluded by the platform.*

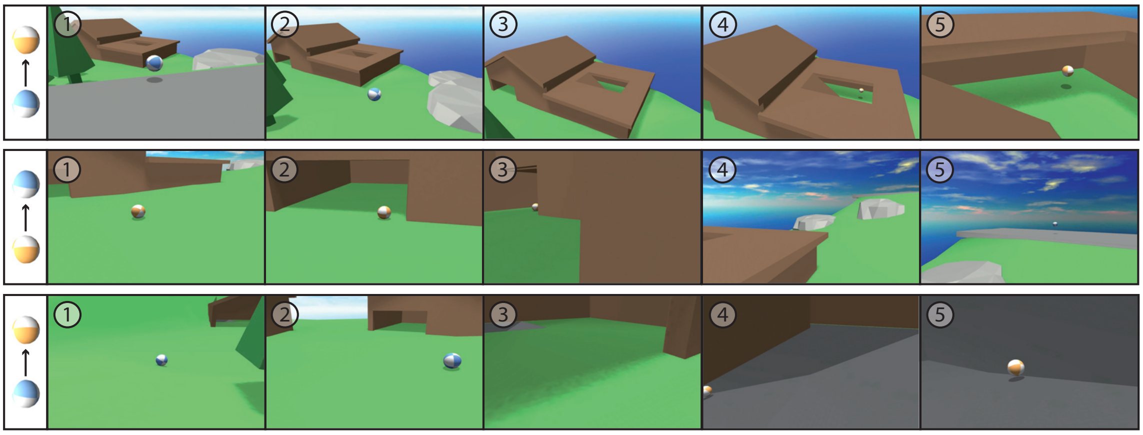


Figure 20: *Three examples of target switching are shown. Our framework allows for completely dynamic start, end, and focus points as input for the planning and is able to switch to another target at any time and get it into view as soon as possible.*

are only complex in two dimensions) to be the best application of our risk prediction. The fast movement of the player's avatar also decreases problems of false alarms, as they are only active during a few frames, and the camera has to move much more to catch up with the player than to react to the false alarm.

4.3 Increasing Awareness of Environment

In the gaming community, it is popular to spectate during game matches. Often, in tournaments where professional players compete against each other, there is a separate camera that shows the action to the crowd. In these events, there is generally a person controlling this spectator camera like a virtual camera man. Automatic camera control, that goes beyond applying standard cinematography rules, are rare. The work of Halper et al. [2003] provides a way to summarize the action in a game and estimate areas of interest. This information can be used to place the camera so that the viewer is shown interesting bits of the match. If the camera is moved to another location, because something interesting has emerged, it needs to be done using a hard cut. These cuts omit all of the space between the two locations, and often, it is of strategic importance to know how players are located with respect to each other.

Our dynamic planning enables target switching, where the camera's focus point switches between multiple focus targets playing at the same time. As the focus dynamically changes between multiple areas or players, our visibility transition planning algorithm results in camera transitions that bring the new player into focus as quickly as possible. The smooth transitions give the viewer a sense of the environment's layout and the distance between the two players, which would not be conveyed if immediate camera cuts were used instead (see Figure 20). This system could complement automatic action summary to show more of the environment, as well as give live spectators another tool for observing the progress of a multi-player game.

5 Performance and Limitations

We have presented an algorithm that enables sophisticated camera transitions by phrasing the problem as a global search on a precomputed visibility-aware roadmap together with local run-time refinement. The balance between pre-computation and run-time calculation allows our system to generate collision-free paths that emphasize visibility while maintaining real-time performance. Because it is global in nature, our method can generate large-scale camera transitions between distant points in a scene, even in complicated geometric situations. In addition, we have demonstrated a variety of extensions to this basic functionality. Our approach can handle dynamic occluders, enhance third-person camera models, perform risk prediction, and respect additional constraints. The computation time is output sensitive. Large camera transitions require tens of milliseconds, while shorter ones used in our third-person camera are computed in one to four milliseconds.

5.1 Performance

Run-time and pre-computation statistics of the environments are given in Table 1. The roadmaps we compute for the different environments have between 819 and 3102 spheres. The largest roadmap, for the *Town* level, requires only 6.36MB. The pre-computation times, the major part of which is computing the sphere visibilities, vary from 8.1 to 61.4 seconds and directly depend on the number of spheres and visibility distance. While pre-computation time is unimportant in many disciplines, it is crucial in the design of interactive environments. Level designers must continually iterate during the design process, testing gameplay with each modification to the level's layout. A long pre-computation time would hinder this iteration, making the camera control system unusable. Thus, our short pre-computation time is critical for a practical system. The average path search and post-processing times depend on the length of the transitions. For the local models used in the *Arena*, this takes 1.8ms on average, while the long transition with dynamic objects demonstrated in the *Tea House* example requires up to 30ms for the initial path. The local model must be extremely fast so that the camera will react adroitly to player movement. However, the longer transitions require several seconds for the camera to travel along the long path (five seconds in the tea house example) and the transition planning is comparatively short. Furthermore, these path computations are performed in a thread parallel to the game engine, which avoids any changes in the frame rate.

Environment	Woodland	Arena	Tea house	Valley	Town
Polygons	1778	15536	8918	8388	34721
Grid resolution	75x25x67	75x14x68	40x6x40	75x21x76	65x21x83
Spheres	2152	2008	819	1106	3102
Roadmap nodes	8730	3991	3999	7953	24989
Visibility distance	100%	30%	100%	100%	25%
Precomputation [sec.]	61.4	27	35	51	28
Memory roadmap [MB]	4.29	2.92	2.07	5.10	6.36
Method	Switching	Local	Dynamic	Dynamic	Transitions
Avg. Path Length [# nodes]	4.8	3.5	12.0	9.6	14.7
Avg. Search [ms]	7.44	1.80	27.67	25.1	14.48
Avg. Post-processing [ms]	2.41	0.41	2.34	6.61	0.63

Table 1: *Roadmap sizes, pre-computation times and run-time performance measurements for the different parts of our algorithm. The environments are depicted in Figure 21 for reference.*

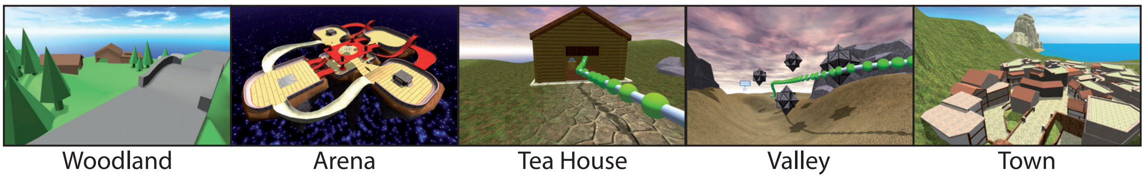


Figure 21: *The different environments used for our experiments.*

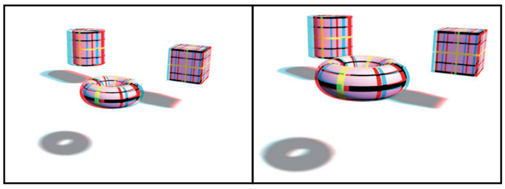
5.2 Limitations and Future Work

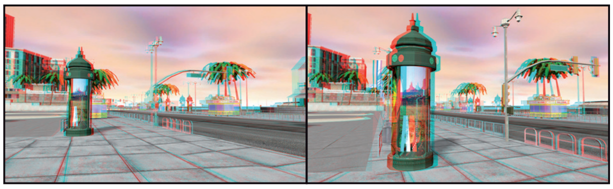
Limitations of our current system direct us to areas of future work. Currently, we support only single-point focus targets. However, one goal is to extend the algorithm to keep an entire object in view by aggregating the visibility estimations for multiple focus targets during the path search, and superimpose the occlusion maps during the refinement phase. Currently, our system computes a fairly dense sampling of the ambient space using the roadmap spheres. Accurately sampling small regions of the environment may require tiny spheres. Such a sampling leads to more accurate visibility estimations, better risk predictions, and superior camera paths. However, extremely dense roadmaps may impede performance. We plan to explore level-of-detail for the visibility-aware roadmap, where *highways* are traveled for large distances, and *local roads* are used for more fine-scale movement.

In this work, we have focused on visibility, since achieving an unoccluded view of an object is the most fundamental goal of camera control. In future research, we wish to include higher-level goals such as cinematographic rules that influence perceptual aspects of the computed transitions to convey different styles or moods. One particularly exciting avenue of research is to generalize from artist-created camera paths during transition planning so that greater control over the character of the transitions is given to the designer.

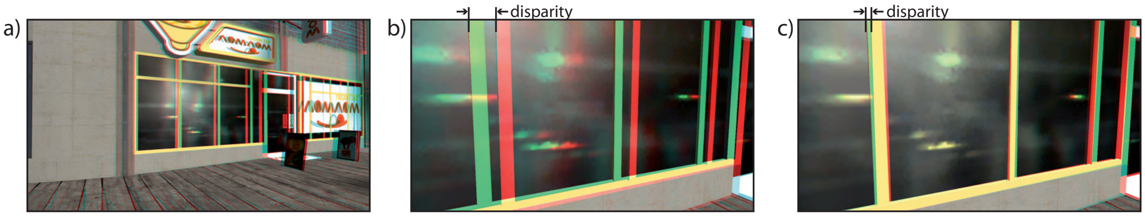
CHAPTER 4

Interactive Stereoscopy





How to control stereoscopy in scenarios where you cannot control the camera position is a delicate task. Restricting the disparities to much will reduce the depth experience, restricting them not enough will create uncomfortable stereoscopy for the viewer. Our optimized stereoscopic camera control is able to handle any situation applying constraints to the perceived depth to get the maximum out of the stereoscopic 3D experience.



Calibrated View:
disparities are within a defined
min-max range

Uncontrolled Stereoscopy:
disparities become too large when
moving towards a wall

Constrained Stereoscopy:
disparities stay in the same range
regardless of scene depth

Figure 1: *This example demonstrates the problem of stereoscopy in interactive applications. a) The image's stereoscopy is calibrated, when the player stands in the scene. b) The image shows what happens with the stereoscopic rendering when the user gets close to a wall. The amount of image disparity becomes very large, causing an uncomfortable viewing situation for the user. c) Controlled stereoscopy, however, is able to keep the disparities within the calibrated min-max range.*

Stereoscopic content creation, processing, and display has become a pivotal element in movies and entertainment, yet the industry is still confronted with various difficult challenges. Recent research has made substantial progress in some of these areas [Lang et al., 2010; Koppal et al., 2011; Didyk et al., 2011; Heinzle et al., 2011]. Most of these works focus on the classical production pipeline, where the consumer views ready-made content that has been optimized in (post-)production to ensure a comfortable stereoscopic experience. See Tekalp et al. [2011] for an overview.

In general, stereoscopy simulates a depth impression by providing a separate image for each the viewer's eyes. Both images show the scene from slightly offset positions, causing the eyes to converge to a certain depth. This convergence is the same as in real life, where each eye perceives objects from slightly different locations. Therefore, stereoscopic images can be created by simply rendering the virtual environment from two viewpoints. The distance of these viewpoints combined with the relative rotation of the two cameras creates images, where each point in the scene is mapped with a horizontal distance within these two images. This difference is called the image disparity, and causes the convergence of the viewers's eyes. In Figure 1, disparities are visualized using red-cyan overlay rendering. The left view is tinted in red while the right view is tinted in cyan.

In reality, eyes have a second mechanism to adjust to different depths, namely the focus depth. Each eye focusses automatically to the depth of the object that the viewer looks at. The change in focus and the convergence of the eyes is naturally coupled. A fundamental problem with stereoscopic 3D shown on current display technology is, that only the convergence can be simulated (through image disparity). The focus of the eyes, however, is always fixed at the screen plane. This creates a de-coupling of the focus and convergence, called the *vergence-accommodation conflict* [Hoffman et al., 2008]. This conflict causes a decrease in visual performance and increases visual fatigue when the convergence point moves away from the focus plane (i.e. the screen plane). Therefore, it is important to keep the perceived depth values within a comfortable range around the screen [Shibata et al., 2011b]. In movie productions, artists often adjust disparities in post-production to strike a balance between perceived depth range and vergence-accommodation conflict.

In interactive applications such as computer games that create stereoscopic output in real-time, one faces a number of fundamentally different challenges [Gateau and Neuman, 2010]. For example, in a first-person game where the player is in control of the view, a simple collision with a wall or another object will result in excessive disparities that cause visual fatigue or destroy stereopsis (see Figure 1).

In order to guarantee proper stereoscopy, one needs a controller that adjusts the range of disparities to the viewer's preferences. An example for such a controller is the work of Lang et al. [2010] which, however, has been designed for post-capture disparity range adaptation using complex image-domain warping techniques. In a game environment where the stereoscopic output is created and displayed in real-time, it is advisable to optimize the stereoscopic rendering parameters, i.e., camera convergence and interaxial separation, and to avoid computationally expensive solutions.

The problem can be formulated as one of controlling perceived depth. We use the term *perceived depth* in the geometrical sense, where the distances reconstructed by the viewer are dominated by the observed screen disparities. Even though there are other important cues such as vertical size or focus that influence perceived depth [Backus et al., 1999; Watt et al., 2005], the work of Held and Banks [2008] showed that the geometrical approach is a valid approximation. The range of perceived depth around the screen that can be viewed comfortably is generally referred to as the *comfort zone*, and is defined as the range of positive and negative disparities that can be comfortably watched by each individual viewer [Smolic et al., 2011; Shibata et al., 2011a]. Therefore, we are looking for an exact mapping of a specific range of distances in the scene into this depth volume around the screen. In the course of this thesis, we will refer to this volume as the *target depth range*. While we concentrate on the control of the mapping between the virtual and real space there exists prior work on how to derive a comfortable target depth range [Woods et al., 1993; Shibata et al., 2011a]

In this chapter, we first discuss the current state of the art in stereoscopy in Section 1. Then, in Section 2, we develop the basics of the stereo geometry to introduce the different terms and their context which we will use throughout this chapter. In Section 3 we show our approach to constraining the camera parameter in order to achieve bounded disparity ranges for any viewing situation and show how to handle changing depth over time in Section 4. Thereafter, in Section 6, we show different applications made possible through our stereo control system before we close with a discussion on the limitations in Section 7.

1 Stereoscopy Background

There has been a lot of research on production and consumption of stereoscopic 3D for many years, with applications ranging from cinema [Lipton, 1982], scientific visualization [Fröhlich et al., 1999], television broadcasting [Meesters et al., 2004; Broberg, 2011] to medical applications [Chan et al., 2005]. A recent survey over the field is provided in Tekalp et al. [2011]. Interestingly, solutions for interactive applications such as games are rare. In the following we discuss related works on stereo geometry, analysis and correction, camera control in real-time environments, and perception.

Stereo geometry: A detailed derivation of the geometry of binocular vision and stereoscopic imaging is given in Woods et al. [1993]. Their main focus is on the description of various image distortions such as keystoning or depth plane curvature, and they show how perceived depth changes under different viewing conditions. Grinberg et al. [1994] also describe the mapping between scene and perceived depth using different frames-of-reference, and propose a framework based on a minimum set of fundamental parameters to describe a 3D-stereoscopic camera and display system. Held and Banks [2008] derive a very complete geometrical model that maps from the scene over the screen to the perceived depth, including the projection to the retina. They not only parameterize the distance from the screen, but also the yaw, pitch, and roll of the viewer's head as well as a relative position to the screen. They use this model to predict distortions perceived by the viewer. Another summary of the stereo geometry is provided by Zilly et al. [2011]. They discuss constraints on the camera separation but do not take the camera convergence into account. Similar to the previous works they are mainly focused on quantifying depth distortions.

In contrast to these works, we provide explicit constraints on both camera convergence and interaxial separation in order to control the mapping of virtual scene content into perceived space. Moreover, none of the previous works has proposed a solution to handle nonlinear visual distortion during temporal interpolation of these parameters.

Stereoscopic content analysis and post-processing: Based on the above works, several methods have been developed for stereoscopic video analysis which estimate image disparities in order to predict and correct visual distortions such as card boarding, the *puppet theater effect*, and other types of distortions [Masaoka et al., 2006; Kim et al., 2008; Pan et al., 2011]. Koppal et al. [2011] describe a framework for viewer-centric stereoscopic editing. They present a sophisticated interface for previewing and post-processing of live action stereoscopic shots, and support measurements in terms of perceived depth. Nonlinear remapping of disparities based on dense depth maps has been discussed by Wang et al. [2008]. Lang et al. [2010] generalize these concepts to more general nonlinear disparity remapping operators and describe an implementation for stereoscopic editing based on image-domain warping. A detailed state-of-the-art report on tools for stereoscopic content production is provided in [Smolic et al., 2011]. This report details challenges in the context of 3D video capturing and briefly discusses the most common problems in practical stereoscopic production such as the comfort zone, lens distortions, etc.

The focus of all these methods is on post-production analysis and correction of stereoscopic live-action video. Our work targets real-time stereoscopic rendering, with control over perceived depth for dynamic 3D environments, minimization of nonlinear depth distortions, and high efficiency for demanding real-time applications. Hence, our goals are complementary to these previous works.

Perceptual research on stereoscopy: An excellent overview of various stereoscopic artifacts such as *Keystone Distortion*, *Puppet Theater Effect*, *Crosstalk*, *Cardboarding*, and the *Shear Effect* and their effects on visual comfort is given in the work of Meesters et al. [2004]. The works from Backus et al. [1999] and Watt et al. [2005] show that perceived depth not only depends on the amount of disparities seen by the viewer, but also on monocular cues such as vertical size, focus, or perspective. The previously mentioned work by Woods et al. [1993] provides a user study to what extent different subjects can still fuse various disparity ranges. The results clearly showed that different persons have significantly varying stereoscopic comfort zones, indicating that individual control over stereoscopic depth is desirable. Stelmach et al. [2003] showed in a user study that shift-image convergence changes are generally preferred over toed-in camera setups, and that static, parallel camera setups are often problematic due to excessive disparities for nearby objects. A perceptual model which emphasizes the importance and utility of individual control over disparity and perceived depth, is described by Didyk et al. [2011]. Shibata et al. [2011a] thoroughly examine the vergence-accommodation conflict and conduct user studies on the comfort of perceived depth for different viewing distances. Their work also provides a way to define a range of disparities that are comfortable to watch by an average viewer.

Results from perceptual experiments and research on stereoscopy clearly indicate a large variation in the physiological capabilities and preferences of different people. These indications motivate the need for tools that allow for a content-, display-, and user-adaptive control of stereoscopic disparity.

Camera control in interactive environments: Finally, there is a large body of work on real-time camera control in virtual 3D environments and games, ranging from intuitive through-the-lens editing interfaces [Gleicher and Witkin, 1992] and cinematographic shot composition [wei He et al., 1996; Bares et al., 1998] to sophisticated camera path planning and minimization of occlusions of points of interest [Oskam et al., 2009]. There exist excellent overviews of this field [Christie et al., 2008; Haigh-Hutchinson, 2009], which address theory of camera control as well as practical solutions for high-performance interactive applications. The work of Jones et al. [2001] also addresses real-time control of camera parameters to adjust the target depth range. However, they assume a parallel camera setup and solve the problem for still images only.

2 Basic Geometric Models of Stereoscopy

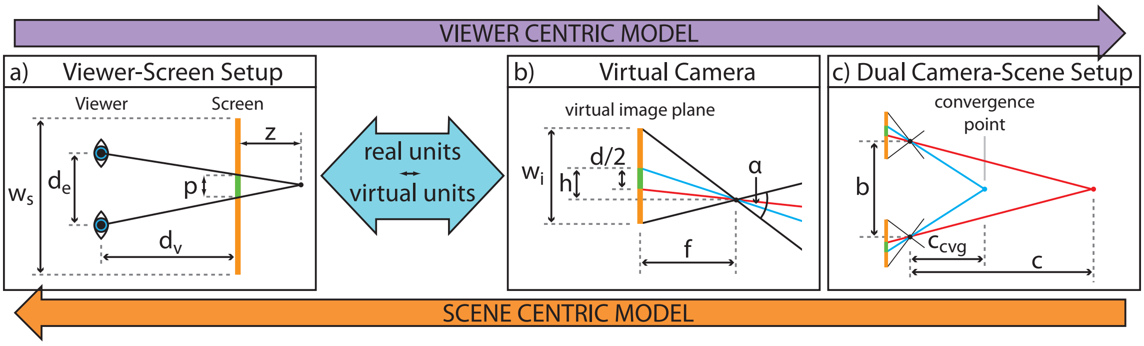


Figure 2: *The geometry pipeline of stereoscopic vision. a) The viewer with eye separation d_e at distance d_v to a screen of width w_s reconstructs a point at depth z in the target space due to the on-screen parallax p. b) The on-screen parallax in a) is caused by a disparity d on the two camera image planes. The camera renders the scene with focal length f and an image shift h. c) Two cameras, with opposite but equidistant image shifts converge at distance c_{cvg} in the scene, and are separated by the interaxial distance b. The image disparity d between both cameras corresponds to a point with distance c from the cameras.*

Depending on the values of several real-world and virtual parameters, the depth perceived by a viewer can dramatically change. Influencing factors are the geometrical configuration of the viewer and the screen plane, the virtual camera model, and the parallel camera configuration used for rendering as well as the scene layout.

In this section we briefly revisit the basic geometry of stereoscopic vision relevant to our work. Our description is, in general, based on previous models [Woods et al., 1993; Held and Banks, 2008; Zilly et al., 2011]. For the stereoscopic camera parameter, i.e. the convergence and interaxial separation, we follow the same definition as the existing models. The interaxial separation b is defined as the distance between the positions of the two cameras. The convergence distance c_{cvg} is defined as the distance between the intersection of the two camera viewing directions and the middle point between the two cameras. Both parameters are schematically shown in Figure 2 c). Note that we converge our cameras using image-shift instead of toeing them in. Image-shift convergence produces less artifacts [Woods et al., 1993; Stelmach et al., 2003].

First, we treat the camera convergence and interaxial separation as unknowns. This will enable us later to derive constraints for these parameters to achieve an optimal mapping of depths between scene- and real-world spaces. Second, we define real-world distances in a 3D volume relative to the screen instead of the viewer. This allows for a more intuitive definition and control of the target depth range (see Figure 2 a). Based on these prerequisites we derive the corresponding *viewer-centric model* and, as the inverse case, the *scene-centric model*, both of which are later needed to formulate constraints for the stereoscopic camera controller and the temporal transformation functions. Figure 2 gives an overview of the geometry pipeline of those two models.

2.1 Viewer-Centric Model

The viewer-centric model describes the reconstructed depth z of a point as a function of the scene distance c, the camera interaxial separation b, and the convergence distance c_{cvg}. This corresponds to a left-to-right traversal of the pipeline in Figure 2.

In order to derive the viewer-centric model, we start with the configuration of viewer and screen. We then describe the conversion from perceived real-world depth to on-screen parallaxes, and convert them to disparities, which are measured in scene units. Finally, we derive the relation of disparities and scene depth values.

Computation of Screen Parallax

Given the configuration in Figure 2 a), where a viewer is looking from a defined distance d_{v} at the screen of a defined width w_{s}. We assume the general setting where the viewer is sitting in front of the screen looking at an orthogonal angle to the center of the screen [Woods et al., 1993]. The viewer is presented with two images, one for each of his eyes. These images contain the same content, but shifted relative to each other. This on-screen difference is called *screen parallax*, which we denote with p. Note, that the distance z assumes positive values when lying behind the screen, and negative values when lying in front of the screen.

The average distance between the eyes of a grown person is about $d_{\mathrm{e}} = 65mm$. The positions of both eyes together with the point perceived at distance z from the screen forms a triangle. This allows to derive the following equality using the *intercept theorem*:

$$\frac{p}{z} = \frac{d_{\mathrm{e}}}{d_{\mathrm{v}} + z}. \tag{1}$$

Solving the equation for z gives

$$z(b, c_{\text{cvg}}, c) = \frac{d_{\text{v}}\, p(b, c_{\text{cvg}}, c)}{d_{\text{e}} - p(b, c_{\text{cvg}}, c)}. \tag{2}$$

We describe the depth z as a function of the camera separation b, camera convergence c_{cvg}, and the in-scene depth c. This way we incorporate into the formula that b, c_{cvg}, and c are unknown parameters, which we seek to constrain later.

The depth value z is positive when lying behind the screen and negative when in front. Similarly, as can be verified by Equation 2, the on-screen parallax p has the same sign as the reconstructed depth z. This explains that points reconstructed by the viewer in front of the screen require the corresponding features in the two images to be shifted with negative distance to each other as opposed to points that are reconstructed behind the screen.

Example: Office Monitor. To show the influence of the different parameters d_{e}, d_{v}, and p, we consider the scenario of a monitor on an office desk. We will use this scenario throughout this chapter to test the effects of changing parameter on the depth mapping. In all cases, the different parameter assume the values listed in Table 1, if not stated otherwise.

d_{e}	6.5	[cm]
d_{v}	70	[cm]
w_{s}	50	[cm]
α	1.0472	[Radians]
f	0.5	[virtual units]
b	2.0	[virtual units]
c_{cvg}	120.0	[virtual units]

Table 1: *Parameter values used in the office monitor examples.*

Figure 3 a) shows the plot of the reconstructed depth z in relation to the parallax p. If the perceived point in space moves towards the position of the viewer ($z = -d_{\text{v}}$), p converges to $-\infty$. On the positive side of the z-axis, p converges towards d_{e}. The parallax crosses the 0-point exactly when z is zero, since there is no parallax introduced for points that lie on the image screen.

As can be seen in Equation 2, the distance between the eyes d_{e} affects the reconstructed depth inversely. This means, that children do perceive greater depth than adults for the same on-screen parallax values. This is an indicator that stereoscopy with large disparities is even more problematic for children.

The perceived depth z, for any given parallax value p, linearly scales with the distance d_{v}, as can be verified in Equation 2. However, the inclination with

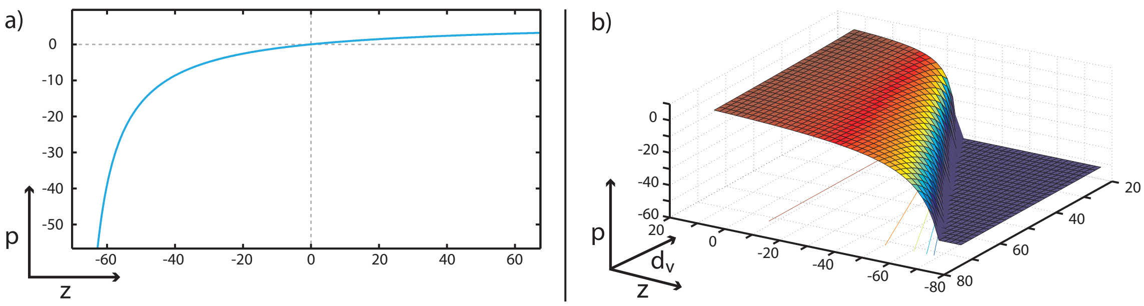


Figure 3: *The relation between the screen parallax p and viewer distance d_v to the reconstructed depth z is shown. a) The reconstructed depth z is shown for changing on-screen parallax values p. It is apparent that the relation is non-linear. b) The parallax p is shown as a function of both the depth z and the viewer distance d_v from the screen.*

which a perceived depth value scales with the distance to the screen depends on the parallax. This means, that each *slice* of z values is scaled differently when changing the viewer distance, and, therefore, the depth range is distorted non-linearly. This effect is shown in Figure 4 a). So, counterintuitively, the perceived depth range increases with the distance to the screen.

Conversion to Image Disparity

Next, we convert the screen parallax p to the image disparity d. The image disparity is the distance between the points of the two images. However, in contrast to the parallax, the image disparity is measured in camera or virtual units, instead of real world measures. Therefore, this conversion is the step where distances are translated from real to virtual. In order to perform the conversion, the ratio between parallax and screen width is multiplied by the width of the virtual image plane.

Given the camera focal length f and the opening angle α, the image width w_i is computed as

$$w_\mathrm{i} = 2f \tan\left(\frac{\alpha}{2}\right). \tag{3}$$

Note, that Equation 3 assumes the opening angle α to be measured without image shift ($h = 0$) in horizontal direction. The measurement of α can be confusing, since the DirectX Framework from Microsoft, for example, defines the opening angle vertically. In this case, Equation 3 computes the image height. To compute the image width, w_i needs to be multiplied by the aspect ratio of the image.

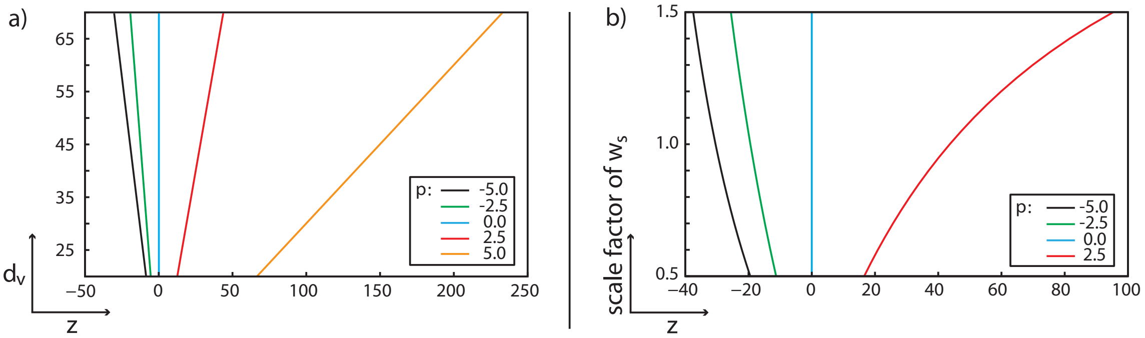


Figure 4: *For different parallax values p, it is shown how the reconstructed depth z changes when changing different parameter. a) The perceived depth z linearly depends on the viewer distance d_v. Although each perceived depth value (for a defined parallax) linearly scales with the distance to the screen, the inclination is different for each value. b) The scaling of the screen width w_s non-linearly influences the reconstructed depth z. Comparing a) and b) it is clear that a scaled screen size cannot be compensated entirely by changing the viewer distance.*

Having computed the image width w_i in camera units, the conversion from parallax p to image disparity d can now be achieved as

$$p(b, c_\mathrm{cvg}, c) = \frac{w_\mathrm{s}}{w_\mathrm{i}} d(b, c_\mathrm{cvg}, c). \tag{4}$$

Inserting Equation 4 into Equation 2, we can derive the relation between image disparities d and the reconstructed depth.

$$\begin{aligned} z(b, c_\mathrm{cvg}, c) &= \frac{d_\mathrm{v} \frac{w_\mathrm{s}}{w_\mathrm{i}} d(b, c_\mathrm{cvg}, c)}{d_\mathrm{e} - \frac{w_\mathrm{s}}{w_\mathrm{i}} d(b, c_\mathrm{cvg}, c)} \\ &= \frac{d_\mathrm{v}\, d(b, c_\mathrm{cvg}, c)}{d_\mathrm{e} \frac{w_\mathrm{i}}{w_\mathrm{s}} - d(b, c_\mathrm{cvg}, c)} \end{aligned} \tag{5}$$

Consider a still stereoscopic 3D image shown on a screen. This implies that the depicted disparity is constant, and, therefore, the parallax is constant as well. Changing the size of the screen will linearly scale the parallax. However, since the on-screen parallax and the perceived depth are non-linearly related, scaling the screen size will ultimately scale the perceived depth range non-linearly. In Section 4, we will further investigate this non-linearity between perceived depth and disparities, and show how this can be taken into account to optimize transformations of the perceived depth over time.

Example: Changing screen size. As stated before, scaling the size of the screen is equivalent to scaling p by the same amount. Let us revisit our office monitor setup. We fix the viewer's position d_{v} as well as the eye distance d_{e}. Figure 4 b) shows the effect of depth perception when scaling the screen. The parallax values p are chosen at the screen scaling of 1.0, and they scale with the screen scaling factor. The figure demonstrates, that the reconstructed depth non-linearly changes with the scaling, with a different curve for each original pair of image features. Comparing this behavior to Figure 4 a), it is apparent that a scaling of the image screen cannot be entirely compensated by changing the view distance. This means, when stereoscopic footage, that was intended for a small TV, is shown on a movie screen, there is no seat in the cinema from which the footage is seen correctly.

The example also shows, that positive parallaxes scale stronger than negative ones. This is because the depth perception, from a geometrical point of view, has higher resolution the larger the parallaxes are. Revisiting Figure 2 a) reveals, that the perceived depth z moves further away for the same increase in p, the closer the parallax comes to d_{e}.

Final In-Scene Distances

Finally, to complete the viewer-centric model, we can compute the distance c in the virtual scene that produces the disparity d, investigating Figure 5. The image disparity d is equally shared between the virtual image planes of the two cameras in the scene. Let us define $\hat{h} = h - d/2$. This equation contracts the image shift and disparity into a single parameter. The geometrical situation with the two virtual cameras looking at a point in the scene at distance c is shown on the right in Figure 5. Two right triangles with the same relative edge length can be spotted: (A, B, C) and (A, B', C'). By dividing the side of each triangle orthogonal to b with the one parallel to b, we get the following equality

$$\frac{\overline{B'C'}}{\overline{AB'}} = \frac{\overline{BC}}{\overline{AB}}. \tag{6}$$

Since the length of each of these edges can be derived from the geometrical configuration in Figure 5, we can rewrite Equation 6 as

$$\frac{f}{\hat{h}} = \frac{c + f}{\frac{b}{2} + \hat{h}}. \tag{7}$$

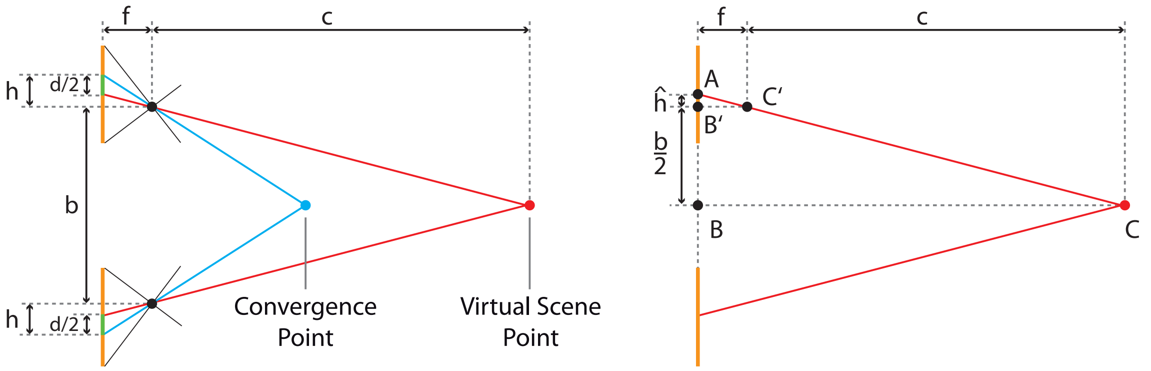


Figure 5: *The geometry of disparity in a virtual camera setup. The disparity d is equally split up between both camera image views. This creates two triangles* (A,B,C) *and* (A,B',C')*, shown on the right, with the same relative edge lengths.*

In order to get the image disparity d as a function of the camera separation b and image shift $h(c_{\mathrm{cvg}})$ as well as the scene depth c, we first need to solve Equation 7 for $\hat{h}$ and then back substitute $\hat{h} = h - d/2$. This gives

$$d(b, c_{\mathrm{cvg}}, c) = 2h(c_{\mathrm{cvg}}) - \frac{fb}{c}. \tag{8}$$

The image shift h is expressed as a function of the camera convergence distance c_{cvg}. Since the proportions between the image shift h and the camera focal length f are the same as between halve the camera separation $b/2$ and the convergence distance c_{cvg}, as can be verified in Figure 2, we can express the image shift as

$$h(c_{\mathrm{cvg}}) = \frac{fb}{2c_{\mathrm{cvg}}}, \tag{9}$$

and finally express the disparity d as

$$d(b, c_{\mathrm{cvg}}, c) = fb \cdot \left(\frac{1}{c_{\mathrm{cvg}}} - \frac{1}{c} \right). \tag{10}$$

Equation 10 describes the image disparity d as a function of camera and scene parameters. By inserting Equation 10 into Equation 5 we can derive

$$\begin{aligned} z(b, c_{\text{cvg}}, c) &= \frac{f\,b\,d_{\text{v}} \cdot \left(\frac{1}{c_{\text{cvg}}} - \frac{1}{c}\right)}{d_{\text{e}}\frac{w_{\text{i}}}{w_{\text{s}}} - f\,b \cdot \left(\frac{1}{c_{\text{cvg}}} - \frac{1}{c}\right)} \\ &= \frac{d_{\text{v}} \cdot \left(\frac{1}{c_{\text{cvg}}} - \frac{1}{c}\right)}{\frac{d_{\text{e}}\,w_{\text{i}}}{w_{\text{s}}\,f\,b} - \left(\frac{1}{c_{\text{cvg}}} - \frac{1}{c}\right)} \\ &= \frac{d_{\text{v}} \cdot (c - c_{\text{cvg}})}{\frac{d_{\text{e}}\,w_{\text{i}}\,c\,c_{\text{cvg}}}{w_{\text{s}}\,f\,b} - (c - c_{\text{cvg}})} \end{aligned} \tag{11}$$

Equation 11 concludes the viewer-centric mapping from the virtual scene to real distances. For any given point in the scene with a distance c fom the line connecting the two virtual cameras, we can now compute the resulting image disparity d (Equation 10), the corresponding on-screen parallax p (Equation 4), and the final distance z from the screen (Equation 11) reconstructed by a viewer.

2.2 Scene-Centric Model

The viewer-centric model allows to predict the reconstructed depth for any point in the virtual scene, given the camera parameters. However, we are also interested in the inverse case, the scene-centric model. Here, the perceived depth z is unknown, and the scene depth c is computed as a function of z and the two stereoscopic parameters b and c_{cvg}.

The scene-centric approach traverses the pipeline in Figure 2 from right to left, effectively inverting each of the steps of the viewer centric model. This also facilitates the derivation of the necessary equations, as we can simply start at the end of the viewer-centric model, and invert the equations.

Similarly to the viewer-centric model before, for the scene-centric model we start with a point in the virtual scene at depth c and want to express it as a function of the camera separation b and convergence c_{cvg}. Equation 10 is the inverse of the equation we want to derive, describing the image disparity d as a function of c and the camera parameter. Solving this equation for c gives

$$c(b, c_{\text{cvg}}, z) = \frac{f\,b}{\frac{f\,b}{c_{\text{cvg}}} - d(z)}. \tag{12}$$

The image disparity d, which depends on the reconstructed distance z, can be scaled back to the on-screen parallax p. Solving Equation 4 for d and plugging it into Equation 12 gives

$$c(b, c_{\mathrm{cvg}}, z) = \frac{f\,b}{\frac{f\,b}{c_{\mathrm{cvg}}} - \frac{w_{\mathrm{i}}}{w_{\mathrm{s}}} p(z)}. \tag{13}$$

The last step to completing the scene-centric model is to express the screen parallax p through the reconstructed depth z and the viewer and eye distance d_{v} and d_{e}. Equation 1 can be rearranged to

$$p(z) = \frac{d_{\mathrm{e}}\,z}{d_{\mathrm{v}} + z}. \tag{14}$$

This function can now be plugged into Equation 13 to finalize the mapping from real space to virtual space as

$$c(b, c_{\mathrm{cvg}}, z) = \frac{f\,b}{\frac{f\,b}{c_{\mathrm{cvg}}} - \frac{w_{\mathrm{i}}}{w_{\mathrm{s}}} \frac{d_{\mathrm{e}}\,z}{d_{\mathrm{v}}+z}}. \tag{15}$$

With Equation 11 and Equation 15 we have related perceived depth and scene distances using the camera convergence and interaxial separation. These equations build the basis for the geometric interpretation of stereoscopy. In the next section, we use both the viewer-centric and scene-centric stereoscopy models to derive constraints on the camera separation b and convergence distance c_{cvg}.

3 Stereoscopic Parameter Constraints

To this end we have discussed the geometric pipeline of stereoscopy and we have derived mappings between real and virtual space that depend on the two basic stereo parameters, namely the camera separation b and convergence distance c_{cvg}. In this section, we examine the influence of b and c_{cvg} on these mappings. First, in Section 3.1, we analyze the standard case where the entire virtual scene depth is mapped into one depth range around the screen. We examine how the perceived depth adjusts when the camera separation b and convergence distance c_{cvg} are changed together and separately. The goal is to derive constraints for the two stereo parameters. Second, in Section 3.2, we examine the general case of constraining multiple scene depth values to defined positions around the screen and verify the constraints found in Section 3.1.

3.1 Standard Case: Mapping a Single Depth Range

The goal is to analyze how the perceived depth changes when the scene is mapped onto a single depth range around the screen. First, we analyze how a given perceived depth $z > 0$ is translated into the virtual scene when changing the camera separation b and convergence distance c_{cvg}. We choose to change the convergence distance c_{cvg} instead of the horizontal image shift h, since it is more intuitive to move the convergence plane directly, than indirectly through h. The image shift h can be computed from c_{cvg} as

$$h = \frac{f\,b}{2\,c_{\text{cvg}}}. \tag{16}$$

Figure 6 a) shows a 2D color-coded plot of the mapping from real to virtual space (Equation 11), where the real depth is fixed to $z = 100$ [cm]. The color in the plot encodes the distance between the mapped value c in the scene and the convergence distance c_{cvg} of the cameras. The relative scenedepth $c_\Delta = c - c_{\text{cvg}}$ is visualized as it corresponds to the real distance (behind the screen = behind the convergence plane c_{cvg}). It is apparent, that c_Δ can geometrically reach beyond infinity, which is, in fact, a mapping to negative infinity or above (mapping towards 0 again from $-\infty$). Additionally, it can be observed that c_Δ converges to $+-\infty$, when the configuration (b, c_{cvg}) moves towards a convergence line. We call this line S_∞.

When the point (b, c_{cvg}) lies exactly on S_∞, the position z is mapped to infinity in the virtual scene. If then either b is decreased, or c_{cvg} is increased, z will be

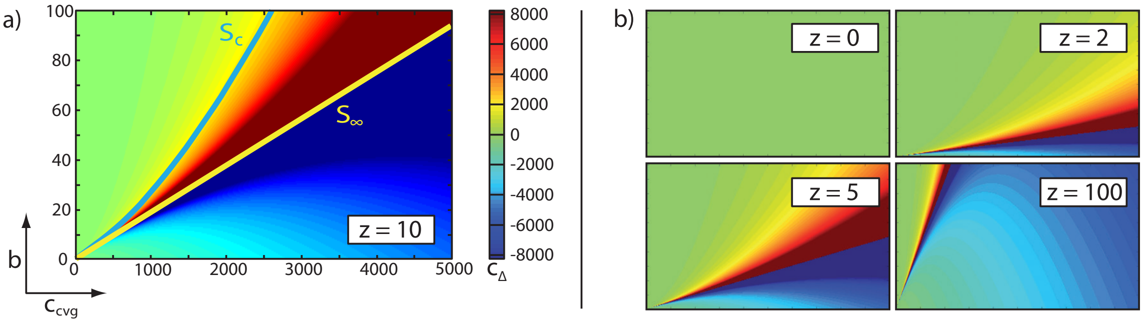


Figure 6: *The behavior of $c_\Delta = c - c_{cvg}$ in a virtual scene when having the real depth z fixed on a value greater than zero, and changing the camera separation b and the convergence distance c_{cvg}. When increasing z, the space of configurations of (b, c_{cvg}), where z is falsely mapped to a large negative value (blue regions), increases as well.*

mapped back into the scene from minus infinity. Observe, that if the point (b, c_{cvg}) lies within the green-red area, there is a one-to-one mapping of the perceived depth z to a scene depth c. If, however, the configuration (b, c_{cvg}) crosses S_∞ (and enters the blue area), z is mapped behind the camera. In this case, there is no point in the scene that appears at distance z to the screen. Figure 6 b) shows, that increased z values cause the space of valid (b, c_{cvg}) configurations to shrink. This means that it becomes harder to find valid stereoscopic parameters for the camera to map further distances into the scene. Therefore, it is desirable to find constraints that automatically bound (b, c_{cvg}).

Let us investigate the mapping from virtual to real space (viewer-centric model) to find a closed form for S_∞. As has been shown before, the line S_∞ is the space where c_Δ converges to infinity. Therefore, the first step is to find a closed expression for c_Δ that depends on the stereoscopic parameter and the real depth z. Starting with Equation 7, we can insert $c = c_\Delta + c_{\text{cvg}}$, and solve for c_Δ, to get the closed form

$$c_\Delta = f\,b \cdot \left(\frac{1}{2h - d} - \frac{1}{2h} \right). \tag{17}$$

The image shift h depends on the camera separation b and convergence distance c_{cvg}, as seen in Equation 16. To represent the value of the real depth z, the image disparity d is used in the equation, as the steps to convert z into d are independent of the other variables in this equation.

Now, in order for $|c_\Delta|$ to become ∞, either of the two denominators on the right side of Equation 17 need to be zero. First, consider $h = 0$. This scenario is equivalent to c_{cvg} being at infinity (parallel cameras). Since we are dealing with values of c_{cvg} less than infinity, we can rule out this case. Second, consider $h = d/2$. This case is only possible with disparities greater than zero as h is always greater than zero. Therefore, the effect of c_Δ approaching ∞ is only possible behind the convergence plane.

Following the second case, S_∞ is thus computed by setting $2h = d_2$, where d_2 is the disparity that produces a fixed perceived depth z_2. Using Equation 16, we obtain

$$S_\infty : b = \frac{d_2}{f} c_{\mathrm{cvg}}. \tag{18}$$

In fact, Equation 18 constrains the camera separation b and convergence distance c_{cvg} together, so that everything in the scene appears no further away than z_2 from the screen. Thus, no matter how the cameras are positioned in the virtual environment, as long as Equation 18 is satisfied, it is impossible to exceed a perceived depth of z_2.

In an interactive application, the content of the scene is known and the maximum depth can be found. It is therefor desirable to constrain the stereoscopic parameters such, that the maximum scene distance c_2, instead of ∞, appears at the real distance z_2. Reformulating Equation 8, the maximum scene depth c_2 can be expressed as

$$c_2 = \frac{f\,b}{2h - d_2}. \tag{19}$$

By replacing h using Equation 16, and solve for b, the general constraint is derived as

$$b = \frac{d_2\, c_2\, c_{\mathrm{cvg}}}{f \cdot (c_2 - c_{\mathrm{cvg}})}. \tag{20}$$

Figure 6 a) shows the curves S_∞ and Equation 20 (denoted S_c) on the landscape of c_Δ for different values of b and c_{cvg}. Note that the curve S_c also has a negative gradient in the c_Δ axis when increasing b or c_{cvg}. This is because we constrain b to $c = c_\Delta + c_{\mathrm{cvg}}$. Since we increase c_{cvg} and keep c constant, c_Δ is decreasing.

We call Equation 20 the *maximum disparity constraint*. The exact value of the convergence distance c_{cvg} is not known at this point. However the constraint ties b to c_{cvg} such, that the maximum disparity always is d_2, no matter how

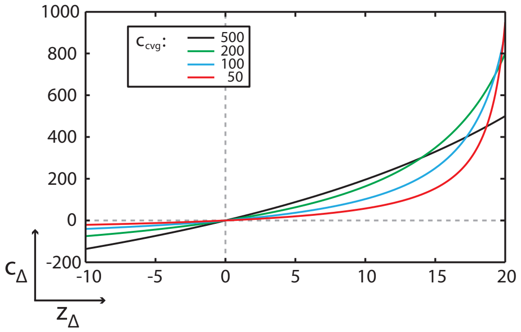


Figure 7: *Mapping of the virtual scene depths $c_\Delta = c_2 - c_{cvg}$ to the perceived depth values z depending on the convergence distance c_{cvg}. The camera separation b is constrained by the maximum disparity constraint (Equation 20) with $c_2 = 1000$. It can be observed, that by changing the convergence distance c_{cvg}, any negative c_Δ can be mapped to any negative z.*

c_{cvg} is chosen. Now, if the *maximum disparity constraint* is fulfilled, there is not only the guarantee to not exceed perceived depths of z_2, but also a full utilization of the depth space behind the screen. Equation 20 makes sure that the furthest point c_2 in the scene is mapped to z_2, and everything in the virtual scene closer to the cameras than c_2 is mapped in front of z_2 in the real space.

So far, only depths of $z > 0$ (with disparities $d > 0$) have been considered. Now, we analyze the second case, where disparities $d < 0$ are investigated, and how they are affected by changing interaxial and convergence distances. The goal is to find a second constraint similar to Equation 20 so that the camera separation b and convergence distance c_{cvg} can be determined uniquely.

The *maximum disparity constraint* ties b to c_{cvg} in such a way that a maximum perceived depth z_2 is never exceeded. However, we have no guarantees yet for an opposing minimum perceived depth z_1. Let us examine how c_Δ behaves for negative disparities, when we constrain b to c_{cvg} (using Equation 20). The behavior of c_Δ for this scenario is shown in Figure 7.

Note that the c_Δ-axis in Figure 7 is relative to c_{cvg}, as $c_\Delta = c - c_{\text{cvg}}$. This means, that the rightmost point of each individual curve in the Figure is mapped to $c_2 - c_{\text{cvg}}$. Analogue, the leftmost point of each curve is also relative to c_{cvg}.

Interestingly, the mapping from z to c_Δ using the *maximum disparity constraint* allows to reach any minimum scene depth c_1 for any given z_1 by constraining c_{cvg}. Thus, the convergence distance c_{cvg} is determined such that the disparity d_1 (corresponding to z_1) maps to the minimum scene distance c_1.

In order to derive this second constraint, we again start at Equation 19. The minimum scene depth c_1 can be expressed as follows

$$c_1 = \frac{f\,b\,c_{\mathrm{cvg}}}{f\,b - d_1\,c_{\mathrm{cvg}}}. \tag{21}$$

Now, the *maximum disparity constraint* (Equation 20) can be used to replace b in Equation 21. Then, solving for the convergence distance c_{cvg}, the second constraint is derived as

$$c_{\mathrm{cvg}} = \frac{c_2\,c_1 \cdot (d_1 - d_2)}{d_1\,c_1 - d_2\,c_2} \tag{22}$$

The form shown in Equation 22 now poses a second constraint on the configuration of (b, c_{cvg}) so that both parameters are uniquely defined. We call this second constraint the *minimum disparity constraint*.

The *minimum* and *maximum disparity constraints* can now be combined to find an exact mapping of any scene content to a predefined perceived depth range. First, the convergence distance c_{cvg} can be computed according to the *minimum disparity constraint* in Equation 22. Then, the camera separation b can be computed using the *maximum disparity constraint* in Equation 20. Useful application scenarios are, for example, mapping of the complete visible scene or of a particular salient object into a prescribed depth range. Another application of these constraints is to adapt to variable z_i boundaries. As the viewer adjusts the desired perceived depth volume, the renderer adjusts the camera convergence and interaxial separation to produce the desired output depth.

3.2 General Case: System of Depth Constraints

In the previous section, we have derived constraints for the camera parameter to achieve an exact mapping of the scene content into a defined range around the screen. In general applications, however, there might be the need for more mapping points. For example, a scene containing an important object, that should also be emphasized in perceived space, while the rest of the scene is still bounded to a certain range. An example is shown in Figure 8. A solution in the general case can be achieved by formulating the mapping problem using multiple constraints, and solving for the best solution in the least-squares sense.

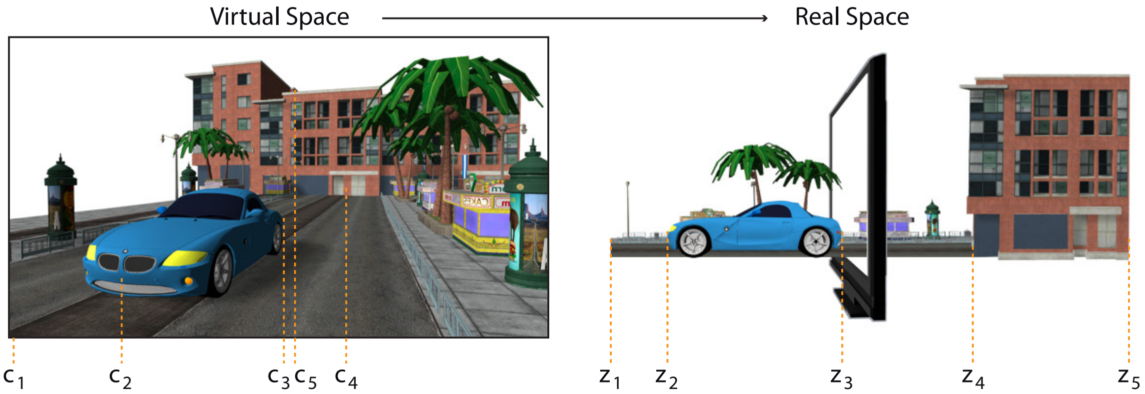


Figure 8: *An example of a general mapping from virtual space to real space is shown. Both the car and the building have a dedicated depth space while the frontmost point of the road should be mapped as close as possible to a specific distance in front of the screen.*

Given a defined series of depth values $[z_1, z_2, \ldots, z_n]$, $z_i < z_j$ for $i < j$, where the screen surface is the reference frame. We want to compute values for the camera separation b and convergence distance c_{cvg} such that a corresponding series of scene points $[c1, c2, \ldots c_n]$, with $c_i < c_j$ for $i < j$, is perceived as close as possible in the least squares sense to the z_i. This will allow us to map salient objects into defined target depth ranges.

First, we use again the relation between perceived depth z and image disparity d in Equation 5, to simplify the problem. The transformation between these two parameters is independent of c_{cvg} and b and, therefore, we can interchange the depth values z_i with the corresponding disparities d_i. The mapping problem can now be formulated by inserting the disparity values into equation Equation 12, and setting them equal to the scene points. This results in the following non-linear system of equations

$$f\,b\,c_i - f\,b\,c_{\text{cvg}} - c_i\,d_i\,c_{\text{cvg}} = 0 \qquad \text{for} \qquad i = 1 \ldots n. \tag{23}$$

To find the optimal values for the camera separation b and convergence distance c_{cvg}, the system of equations in (23) can, for example, be solved in a least-squares sense with a Gauss-Newton solver.

Verification of minimum and maximum disparity constraints The special case using two real-world depth values $[z_1, z_2]$ and two scene points $[c_1, c_2]$ can be computed analytically, as there remain two equations for two unknowns. In this case the above system has one non-trivial solution, and we can analytically determine the constraints for c_{cvg} and b. We start out with two equations and two unknowns (b and c_{cvg}) in the system. The equations are written as

$$f\,b\,c_1 - f\,b\,c_{\mathrm{cvg}} - c_1\,d_1\,c_{\mathrm{cvg}} = 0 \tag{24}$$

$$f\,b\,c_2 - f\,b\,c_{\mathrm{cvg}} - c_2\,d_2\,c_{\mathrm{cvg}} = 0. \tag{25}$$

First, Equation 25 can be solved for b

$$\begin{aligned} f\,b\,c_1 - f\,b\,c_{\mathrm{cvg}} &= c_1\,d_1\,c_{\mathrm{cvg}} \\ f\,b\,(c_1 - c_{\mathrm{cvg}}) &= c_1\,d_1\,c_{\mathrm{cvg}} \\ b &= \frac{c_1\,d_1\,c_{\mathrm{cvg}}}{f\,(c_1 - c_{\mathrm{cvg}})}. \end{aligned} \tag{26}$$

Now, the expression in Equation 26 can be used to remove b from Equation 25

$$\frac{c_1\,d_1\,c_{\mathrm{cvg}}}{(c_1 - c_{\mathrm{cvg}})}\,c_2 - \frac{c_1\,d_1\,c_{\mathrm{cvg}}}{(c_1 - c_{\mathrm{cvg}})}\,c_{\mathrm{cvg}} - c_2\,d_2\,c_{\mathrm{cvg}} = 0. \tag{27}$$

At this point, the focal length f is already cancelled out. Now, Equation 27 can be solved for the analytic solution of c_{cvg}

$$\begin{aligned} c_1\,d_1\,c_2\,c_{\mathrm{cvg}} - c_1\,d_1\,c_{\mathrm{cvg}}\,c_{\mathrm{cvg}} - c_2\,d_2\,c_{\mathrm{cvg}}\,(c_1 - c_{\mathrm{cvg}}) &= 0 \\ c_1\,d_1\,c_2\,c_{\mathrm{cvg}} - c_1\,d_1\,c_{\mathrm{cvg}}^2 - c_1\,c_2\,d_2\,c_{\mathrm{cvg}} + c_2\,d_2\,c_{\mathrm{cvg}}^2 &= 0 \\ c_{\mathrm{cvg}}^2\,(c_2\,d_2 - c_1\,d_1) + c_{\mathrm{cvg}}\,c_1\,c_2\,(d_1 - d_2) &= 0 \\ c_{\mathrm{cvg}}\,(c_2\,d_2 - c_1\,d_1) &= c_1\,c_2\,(d_2 - d_1) \\ c_{\mathrm{cvg}} = \frac{c_1\,c_2\,(d_1 - d_2)}{c_1\,d_1 - c_2\,d_2}. \end{aligned} \tag{28}$$

Equation 28 is the formula that computes analytic solution for the camera convergence c_{cvg} depending on the scene depths c_1 and c_1 and the disparity values d_1 and d_2. It can be veryfied that Equation 28 is, in fact, the *minimum disparity constraint* (Equation 22).

Now, Equation 28 can be back substituted into Equation 26, to get the analytic solution for the camera separation b independent of c_{cvg}.

$$\begin{aligned}
b &= \frac{c_1 d_1 \frac{c_1 c_2 (d_1 - d_2)}{c_{\text{cvg}} (c_1 d_1 - c_2 d_2)}}{f (c_1 - \frac{c_1 c_2 (d_1 - d_2)}{c_{\text{cvg}} (c_1 d_1 - c_2 d_2)})} \\
&= \frac{c_1 d_1 c_1 c2 (d_1 - d_2)}{f (c_1 (c_1 d_1 - c_2 d_2) - c_1 c_2 (d_1 - d_2)} \\
&= \frac{c_1 d_1 c_1 c2 (d_1 - d_2)}{f (c_1 c_1 d_1 - c_1 c_2 d_2 - c_1 c_2 d_1 + c_1 c_2 d_2} \\
&= \frac{c_1 d_1 c_1 c_2 (d_1 - d_2)}{f c_1 d_1 (c_1 - c_2)} \\
&= \frac{c_1 c_2 (d_1 - d_2)}{f (c_1 - c_2)}. \qquad (29)
\end{aligned}$$

Equation 29 is the independent constraint for b when mapping one virtual depth range exactly onto one depth range in real space. It can be veryfied that Equation 29 can be derived by inserting the *minimum disparity constraint* in Equation 22 into the *maximum disparity constraint* in Equation 20.

For both the standard case where one range in virtual space is mapped onto one range in the real space, and the general mapping of arbitrary points, an optimal solution in the least squares sense can be found for the two stereoscopic parameters b and c_{cvg}. To this end, the constraints consider a static scene depth, where no movement of the camera or the objects in the scene occur. However, such movement introduces new problems, as the per-frame constraining of the stereoscopic parameters can cause discontinuous perceived depth over time. In the following section we discuss this problem in more detail and propose a temporal parameter interpolation that minimizes distortions in the perceived space.

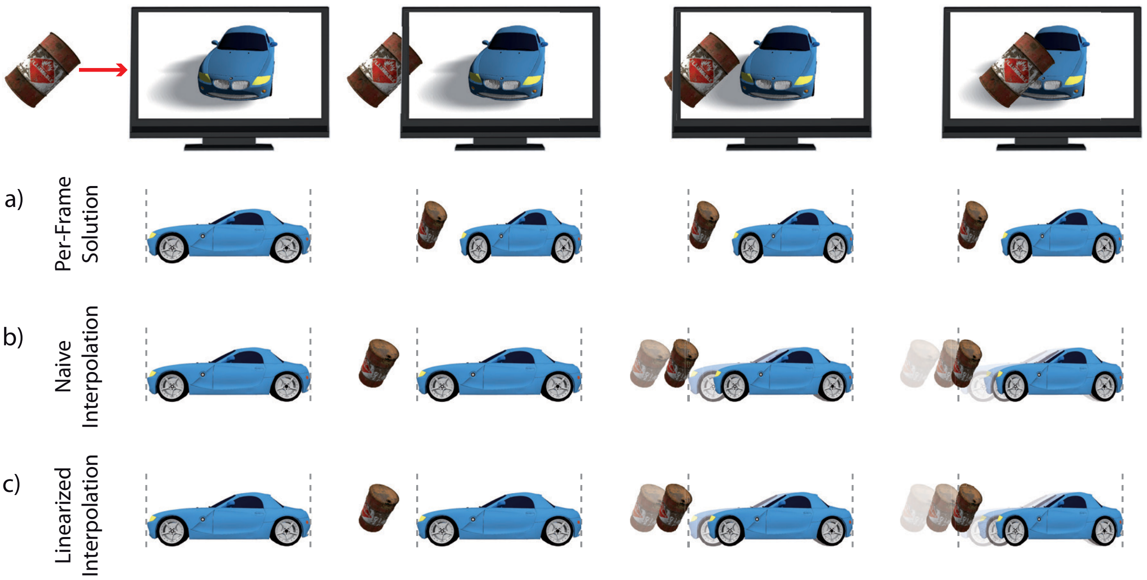


Figure 9: *The problem of discontinuous depth change when an object enters the scene. The top row schematically demonstrates an example where a barrel enters the view frustum from the left at close proximity. Below, different ways to handle this situation are shown in real space. a) The per-frame solution causes an instant rescaling of the scene, which is noticeable and undesired. b) Interpolating the camera separation b and convergence distance c_{cvg} provides a smooth solution, but the transformation of the perceived space is non-linear. c) We propose to perform a linearized transformation in perceived space to provide a smooth and controlled depth adaptation over time.*

4 Temporal Constraint Interpolation

In the previous section, we have discussed how the basic stereoscopic parameters can be constrained in order to keep the perceived depth range within a defined limit. The constraints, however, only consider a snap-shot in time. In an interactive environment, unpredictable object- or viewer-motion can change the scene depth instantly. This causes two problems. Let (b^t, c^t_{cvg}) denote the set of stereoscopic parameters at time t. On the one hand, if the scene depth changes from time $t-1$ to t and the interaxial separation and convergence distance are kept constant at $(b^{t-1}, c^{t-1}_{\text{cvg}})$, the scene is mapped to a different target range, which can result in excessive disparities and compromise the stereoscopic perception. On the other hand, if the camera convergence and interaxial separation are immediately re-computed as (c^t_{cvg}, b^t) according to the constraints introduced in the previous section, the new c^t_i are again mapped to the original depth volume. However, the perceived depth of scene elements visible at both time-steps will change instantly, as shown

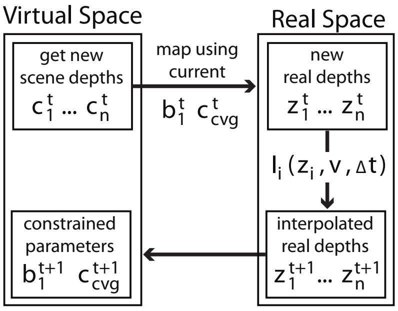


Figure 10: *Overview of our proposed temporal parameter interpolation. Instead of interpolating the camera separation b and convergence distance c_{cvg} directly in virtual space, we propose to compute depth transformations into the real space. This allows the application of desired transformation functions I_i to the real depth values z_i. Then, using the interpolated depths, the new stereo parameters, that achieve this desired depth change, can be computed using our constraining.*

in Figure 9 a). These sudden jumps in depth can be irritating to the viewer as well. So in general, we would like to control the stereoscopic parameters over time in order to reduce both types of artifacts.

The straight forward solution would be to interpolate linearly between the two parameter sets $(b^{t-1}, c^{t-1}_{\text{cvg}})$ and (b^t, c^t_{cvg}). However, a simple linear change of camera convergence and interaxial separation causes the target depth range to transform in a nonlinear fashion, as shown in Figure 9 b). This scaling of the target depth results in nonlinear changes of object shapes and of the scene volume over time. The non-linearity depends on the entire range of parameter that are used in the transformation from virtual to real space. It is therefore difficult to predict the transformation, even if it is smooth.

In order to minimize these types of visual artifacts, we derive an interpolation function for our stereoscopic constraints that linearizes changes in perceived depth while keeping the perceived depth volume approximately constant. An overview of this approach is shown in Figure 10. Let z_i^t denote a depth value at time t in target space with respect to the screen. In order to keep the perceived depth volume constant over time, we can define an arbitrary (not necessarily linear) interpolation function $I_i(z_i^t, z_i^{t-1}, \kappa)$ for each of the depth values z_i^t. Each function gradually changes the point z_i^t back to its position z_i^{t-1} at a previous time $t-1$. In order to control all interpolation functions for all individual points simultaneously, we define the interpolation variable κ as a function that depends on the current time step Δt of the interpolation and a predefined velocity v that is used to control how fast the target depth range transforms

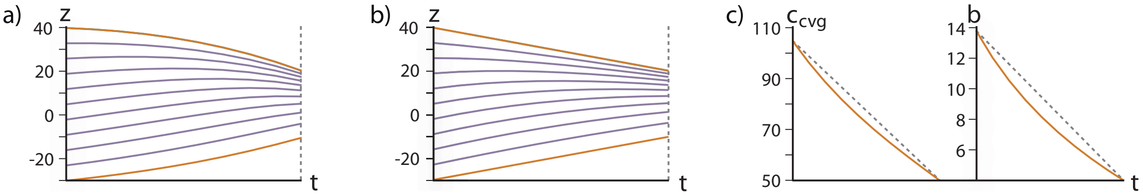


Figure 11: *The temporal interpolation of camera convergence and interaxial separation is demonstrated. a) An exxample how the target depth range transforms over time when the stereoscopic parameters are linearly interpolated is shown. b) The same range is interpolated as in a), using our linearized transformation. c) The two graphs show the functions of c_{cvg} and b that achieve the linearized range transformation in b).*

$$\alpha(\Delta t, v) = \min\left(\frac{v\,\Delta t}{\frac{1}{n}\sum_{i=1}^{n}\operatorname{len}(I_i)}, 1\right). \tag{30}$$

The value of κ is computed as the ratio between the distance for the time step Δt and the average length of all the control curves I_i. We use the min function in Equation 30 to prevent *overshooting* of the interpolation in case of a large time-step Δt or velocity v.

Now, in order to keep the target depths approximately constant over time, as soon as the scene depth values change from c_i^{t-1} to c_i^t, we first compute the resulting new target depths z_i^t, and then use the individual depth interpolation functions I_i to gradually update the camera convergence and interaxial separation to restore the target depths back to z_i^{t-1}.

Linearized range interpolation. Similar to Section 3, the special case of interpolating between two depth ranges defined just by their respective minimum and maximum depth is of particular interest. Using our above formulations, we can define the interpolation in terms of the z_i^t , which allows us to linearize the change in target depth. Let $[z_1^{t-1}, z_2^{t-1}]$ and $[z_1^t, z_2^t]$ define the two depth ranges. Then the standard linear interpolation functions I_{lin} on the range boundaries achieve the desired linear change in depth

$$I_{\text{lin}}(z_i^t, z_i^{t-1}, \kappa) = (1-\kappa)\, z_i^t + \alpha\, z_i^{t-1}. \tag{31}$$

The graphs in Figure 11 show the effect of Equation 31 on the target depth range over time. Figure 11 a) shows how the depth range from $[40, -30]$ cm is transformed to $[20, -10]$ when the camera separation b and convergence distance c_{cvg} are linearly interpolated. Figure 11 b) shows our linearized approach. The transformation is linear along the boundaries of the target depth range. Compared to a simple linear interpolation of b and c_{cvg} , our linearized transformation introduces significantly less distortions over time.

The resulting functions for b and c_{cvg} over time for the linearized transformation are shown in Figure 11 c). In both graphs, the dashed line shows the linear interpolation of the parameter. It is apparent that our optimized functions differ considerably from linear interpolation of the parameters.

Algorithm 1: *Stereoscopic Parameter Update*

```
1: procedure UpdateStereoParameter($c_{\text{cvg}}^{t-1}, b^{t-1}$)
2:     $[c_1^t, c_2^t] \leftarrow$ getSceneDepthRange()
3:     $[z_1^t, z_2^t] \leftarrow$ mapDepth($c_1^t, c_2^t, c_{\text{cvg}}^{t-1}, b^{t-1}$)
4:     $\kappa \leftarrow$ computeAlpha($\Delta t, v$)
5:     $[d_1, d_2] \leftarrow$ transform($I_{\text{lin}}, z_1^t, z_2^t, \kappa$)
6:     $[b^t, c_{\text{cvg}}^t] \leftarrow$ constrainParameter($c_1^t, c_2^t, d_1, d_2$)
7:     return $[c_{\text{cvg}}^t, b^t]$
8: end procedure
```

5 GPU Implementation

Our stereoscopic controller algorithm produces a sequence of parameter pairs (c_{cvg}^t, b^t) at each frame t. Pseudo code of the parameter update is provided in Algorithm 1.

After the scene depth distribution has changed, possibly in a discontinuous fashion, the first step is to acquire the new scene depth range $[c_1^t, c_2^t]$ (line 2). Efficient computation of this range is described below. Using this range, the new real depth range $[z_1^t, z_2^t]$ caused by the new scene depth values in this frame can be computed (line 3) using Equation 11. Given the target depth range $[z_1, z_2]$, and the velocity v, the interpolation variable α can be determined (line 4) with Equation 30. Then, the target depth range can be interpolated over time (line 5) using the interpolation function in Equation 31 converted to the corresponding disparity values $[d_1, d_2]$ (using Equation 5). Finally, using the stereoscopic parameter constraints, Equation 20 and Equation 22, the new values for c_{cvg}^t and b^t can be computed (line 6).

Efficient depth range acquisition. In real-time applications, budgets are tight and efficiency is a critical factor. To be usable in a production scenario, any real-time algorithm needs to fit into a budget of a few milliseconds (at 60fps, the total frame budget is only 16.7ms). The only stage of our algorithm that cannot be computed in constant cost is the determination of the minimum and maximum depths in the scene. To efficiently obtain these depths, we perform a number of min-max reduction passes of the depth buffer on the GPU [Greß et al., 2006].

The idea of min-max reduction is to divide the search for the scene minimum and maximum depth values to utilize the parallel nature of the CPU. Figure 12 shows how this is performed. In the first pass, the depth buffer is reduced

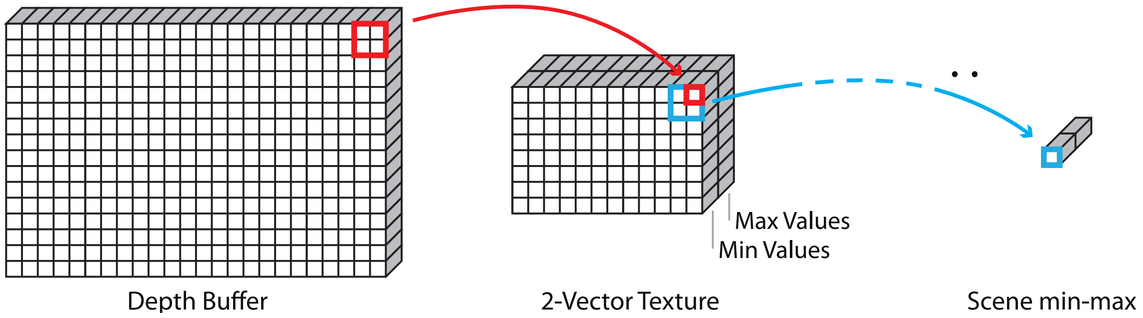


Figure 12: *The min-max reduction to efficiently determine the scene minimum and maximum depths. In the first pass, the depth buffer is reduced by finding the minimum and maximum depth values for each 2-by-2 patch and storing the extrema in a 2-vector texture. All subsequent passes reduce the texture each 2-by-2 patch at a time until only a 1-by-1 texture remains. This texture contains the scene maximum and minimum depth.*

by finding the extrema within each 2-by-2 patch. The resulting min-max values are stored in a 2-vector texture, where each texel contains the values computed from the previous 2-by-2 patch. In all subsequent steps, the texture is reduced each 2-by-2 patch at a time until only a 1-by-1 texture remains. This texture contains the minimum and maximum depth values that occur in the scene in the current frame. Although the number of passes is logarithmic in the screen size, modern GPUs are highly optimized for this workload and these passes are computed very cheaply. We timed the passes on a recent Nvidia GPU (the GTX580) and the cost was only 0.09ms at 1280x720 and 0.18ms at 1920x1080.

Once these depths are obtained they can either be transferred back to the CPU for processing or processed directly on the GPU. We choose the latter option as GPU to CPU communication can result in GPU stalls on current architectures. To this end, the interaxial and convergence distance are stored in a 2 texel floating point texture. Algorithm 1 is performed in a shader that reads the min-max depth and previous frame stereoscopic parameters from a 2 texel texture and writes a new set of stereoscopic parameters to a second 2 texel texture. At each iteration the role of the textures are switched. In this way, the stereoscopic parameters will always be available in a GPU buffer as input to the stereoscopic render engine.

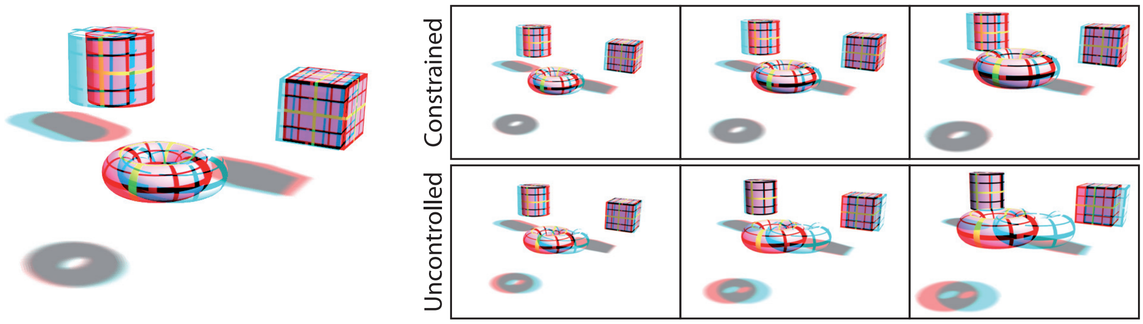


Figure 13: *Automatic stereoscopic control demonstrated on a simple example. The image on the left shows the scene with properly calibrated stereoscopic parameters. The torus is reconstructed in front of the screen while the cylinder is perceived behind the screen. While the camera moves towards the objects, our constraints adjusts the stereoscopic parameters such that the torus remains in front of the screen and the cylinder behind the screen. Uncontrolled stereoscopy, in contrast, causes the scene to shift towards the viewer and cause exceeding disparities on the torus, while the cylinder is moved to the zero disparity plane (perceived at the screen).*

6 Results and Evaluation

To this end we have presented a system to constrain the two basic stereoscopic render parameters, the camera separation and convergence distance. Additionally, we have discussed how to resolve temporal depth discontinuities. In the following we present stereoscopic images generated using our controller and discuss several areas of application.

6.1 Adaptive stereoscopy and automatic fail-safe

When moving the camera through a scene, the render-depth is constantly changing. Improper camera convergence and interaxial separation can cause large disparity values when an obstacle suddenly comes close to the camera. The concept is shown in Figure 13. On the left, the scene is shown with properly calibrated stereoscopic parameters. On the right, a comparison between stereoscopy using our constraints and uncontrolled stereoscopy is shown for the camera moving towards the objects in the scene. While our controller assures that the disparities stay in the same range, image disparities grow extremely large in the uncontrolled case. Also, positive disparities (on the cylinder in the back of the scene) that correspond to a reconstructed depth behind the screen are decreased to practically zero for uncontrolled stereoscopy. This shift of the perceived depth range is unintentional and

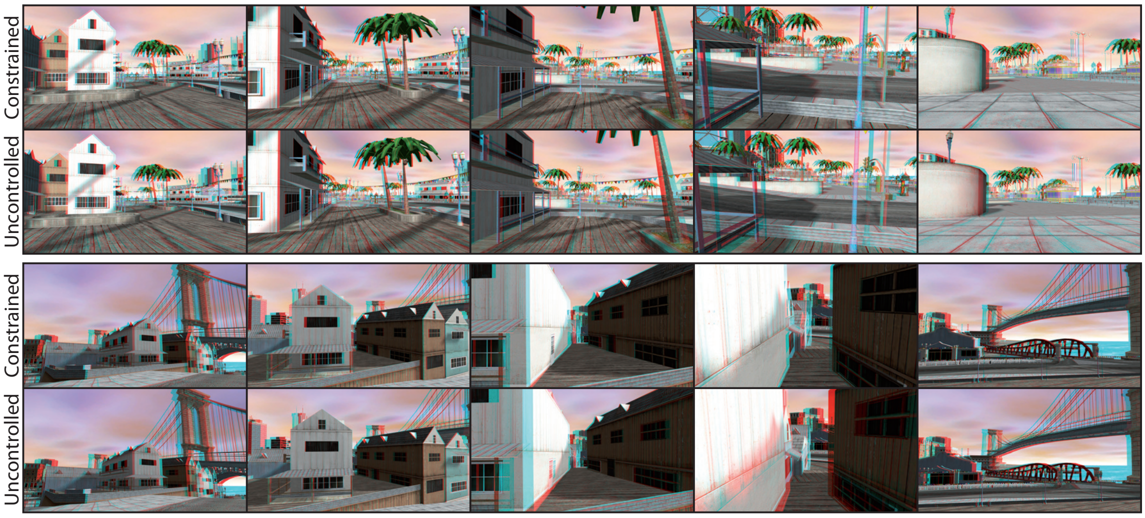


Figure 14: *Two comparisons between constrained and uncontrolled stereoscopy of medium to fast camera motion through complex environments. At the beginning of the shot, the uncontrolled camera has the exact same parameter calibration as our controller, so that initially comfortable stereopsis is ensured. The uncontrolled camera fails to preserve a comfortable disparity range, causing excessive disparities and hence inducing eye-strain to the viewer.*

reduces the positive perceived depth (depth behind the screen). In contrast, our parameter constraints also maintain the depth contrast behind the screen, effectively utilizing the depth budged available.

The automatic nature of our stereo controller is especially useful in computer games. Here, the user has usually full control over the camera, and can freely move through the environment. Restricting this movement will also restrict the players freedom to explore the environment. This creates the problem that a player might move the camera, that might render properly calibrated stereo in one moment, towards obstacles or walls. This causes an unpredictable variability in the depth ranges displayed within a short period of time. Two examples are shown in Figure 14, where a player is moving through the environment. While our method is able to adapt to vast depth changes, uncontrolled stereoscopy causes excessive disparities and hence induces eye-strain to the viewer.

In real-time engines for simulations and games, objects are usually easily trackable. They can be assigned labels by artists or given some importance based on the configuration in the scene. These labels can be utilized to enforce constraints on the placement of important scene objects in the reconstructed space around the screen. Figure 15 shows an example of our parameter optimization, mapping multiple points in the scene onto multiple target ranges.

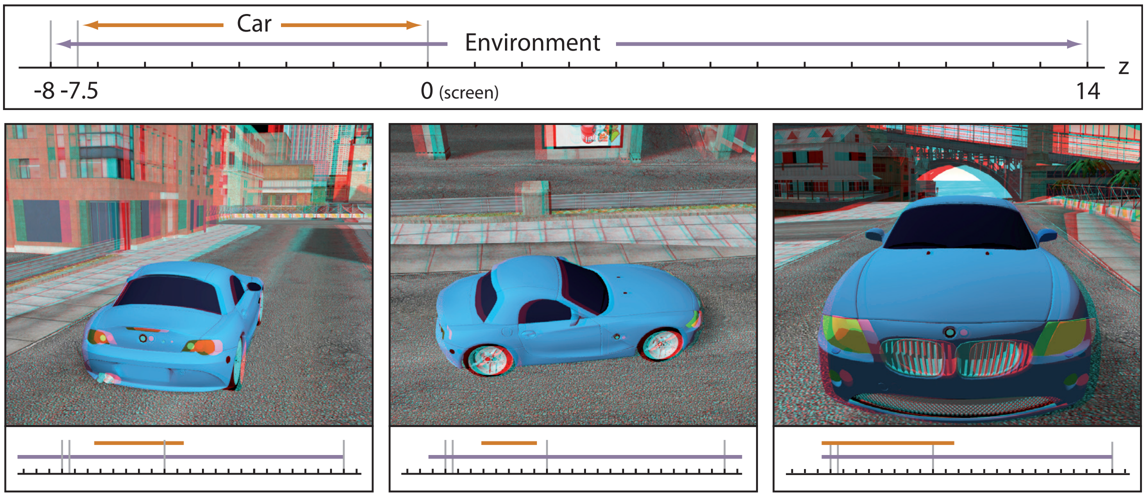


Figure 15: *Example of a scene with multiple points mapped onto multiple target depth values. The top row shows the desired mapping. The car should appear in the target range [-7.5, 0.0] cm while the environment should be mapped into [-8.0, 14.0] cm. The bottom row shows how different views are rendered as close as possible to the desired mapping in the least squares sense using our nonlinear optimization.*

The top graph shows the desired mapping of the car and the environment in real space. The bottom images show stereoscopic renderings of the least-squares solution for the stereoscopic parameters and the mappings of the car and environment. The goal is to have the entire scene fit in a certain depth range and place the car just in front of the screen. While the scene depths change between long range views (first and third image in Figure 15) and a close up shot (second image in Figure 15), the least squares solution keeps the car in front of the screen and the scene from causing exceedingly large disparities.

Another typical situation encountered in interactive, dynamic environments is the sudden change of scene depth. An example is shown in Figure 16, where the camera is horizontally translated across the street. It passes very closely in front of a couple, creating a sudden discontinuous decrease in the minimum scene depth. If this is not handled properly, the depth perception is immediately destroyed. The graphs in Figure 16 show how our method adapts the stereoscopic parameters over time to prevent exceeding disparities to appear for too long.

Wile our stereoscopic controller is light-weight and robust enough to be active at all times, it also can be regarded as a safeguard that adjusts the stereoscopic parameters only when disparity thresholds are surpassed. Most

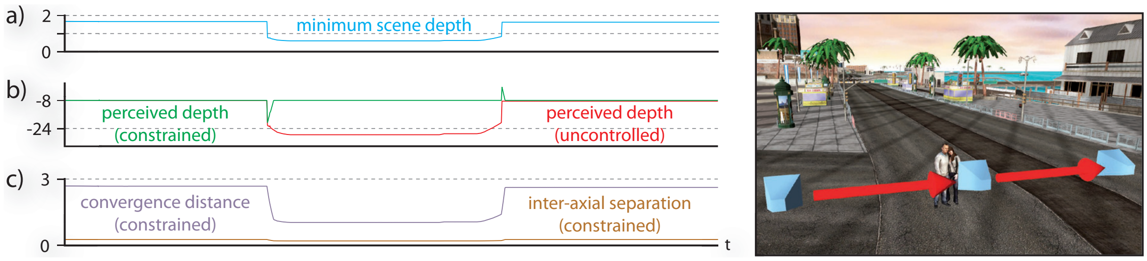


Figure 16: *Comparison between constrained and unconstrained stereo cameras while horizontally moving the camera past a close obstacle. a) The minimum scene depth over time is shown. b) The depths reconstructed by the viewer are shown. Unconstrained stereoscopy (red) causes a far too low perceived depth. The constrained camera adapts smoothly to the discontinuous depth (green). c) The changes to the stereoscopic parameter computed using our controller are shown.*

interactive 3D applications have a defined view that is more or less the same most of the time. An example is a racing game, where the camera generally follows the car and mostly hast a similar view onto the track ahead. In such a setting it might be desirable to use a set of tuned stereoscopic parameters that display the setup properly in an artist defined depth range. Using the viewer-centric model (Section 2.1) the depth values that the scene causes the player to perceive can be monitored with little overhead. Should something in the scene, like an object flying towards the camera, cause exceeding depth values, our controller can simply be switched on and use the constraints to keep disparities within defined thresholds until the situation is resolved.

6.2 Artist Control and Production

Our method provides intuitive and exact stereoscopic control by allowing the viewer to adapt the perceived borders of the depth range to the respective personal comfort zone. The user can also define high-level goals for the depth, as shown in Figure 17. The viewer may specify to move or scale the target depth image, without worrying about the exact depth values or the stereoscopic camera parameters.

Our disparity constraining also enables novel tools for content creators and artists in production. Our controller can be used to intuitively script perceived depth for different scene parts and hence create artistic effects such as emphasized depth for certain objects, stereoscopic flatlands, etc. This gives the camera artists a tool to adjust the stereoscopic depth perception without actually knowing the depth of the scene. By placing hit-boxes and defining a

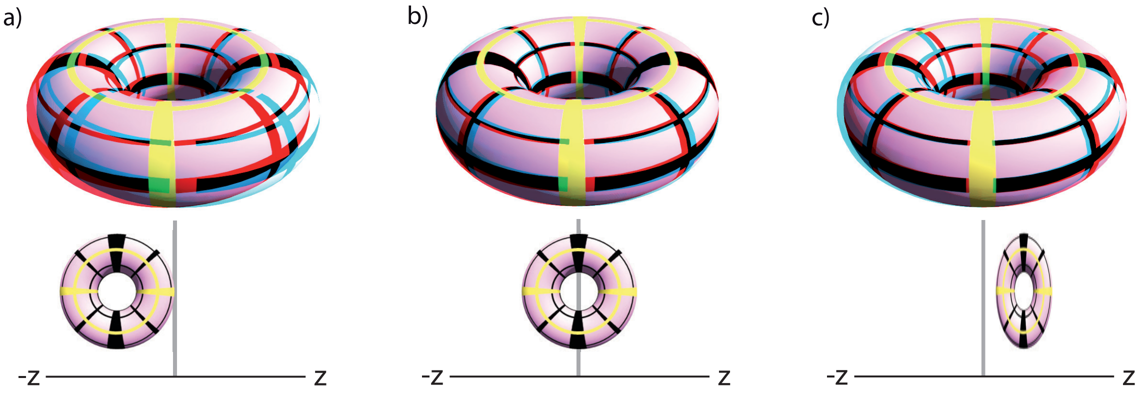


Figure 17: *Examples of exact stereoscopic control using our method. a) The torus is rendered such that it appears directly in front of the screen plane. b) Exactly half of the torus appears in front and the other half behind the screen. c) The torus appears one seventh of its original target length behind the screen and its perceived length is halved. With our controller such settings can be guaranteed while the viewpoint changes dynamically.*

constrained depth range, the artist knows what depth will be rendered regardless of the scene configuration. The artist can also key-frame the behavior of the depth range to react to events. An example would be to simulate pressure waves of an explosion by adding a wiggling to the perceived depth zone. Another example, shown in Figure 18, is the control of the perceived depth in advanced camera effects. The figure shows a comparison of controlled versus uncontrolled depth for the vertigo shot, where the camera focal length is changed over time to create a dramatic effect. Since the perceived depth also depends on the camera's focal length, the depth range changes with the parameter. For an artist, it is very hard to control this. However, using our stereoscopic controller, the depth range can remain constant, or, if the artist desires, be key-framed.

	TV	PC	3DS
TV	[-51.4, 86.5]	[-6.1, 8.2]	[-0.5, 0.6]
PC	[-65.8, 164.7]	[-8.0, 14.0]	[-0.7, 1.0]
3DS	[-837.1, 3532.2]	[-105.4, 240.9]	[-1.0, 1.5]

Table 2: *Content scaling matrix. The entry in the row i and column j shows the target depth range (in cm) of the image that is produced for i and viewed on j. The viewing conditions for each device (in cm): TV: $w_s = 100, d_v = 300$. PC: $w_s = 50, d_v = 65$. Nintendo 3DS: $w_s = 7.6, d_v = 35$.*

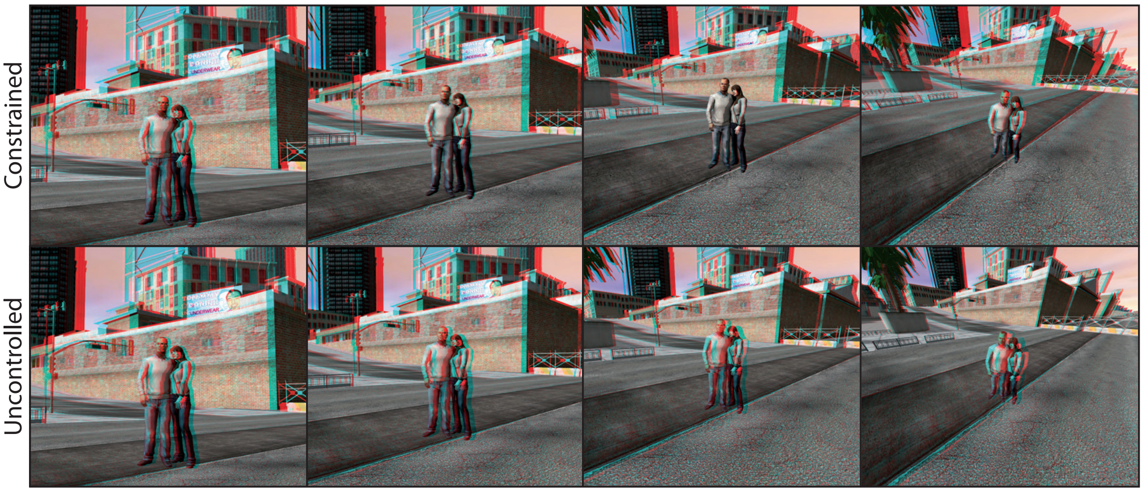


Figure 18: *An example of aided artist control for advanced effects. For this vertigo shot, the artist is able to just define the depth range and perform the effect without worrying about the change in perceived depth. The uncontrolled case shows that changing the camera focal length results in the depth range shifting our of control.*

Since our stereoscopic controller takes the entire viewing situation into account (i.e. viewer distance, screen size, etc.) all the above benefits apply not only to the screen that the content is produced for, but also allows to apply the content to any screen size. Stereoscopic images that are produced for a certain target screen and viewing distance are optimized to create a particular depth impression. If, however, the content is shown under different viewing conditions, the viewer may experience a completely different depth sensation. Figure 19 shows a comparison of two different views that are optimized for a *Television Screen*, a *PC Monitor*, and *Nintendo 3DS*. The viewing conditions and target depth ranges for each device can be found in Table 2. The stereoscopic image created for a Nintendo 3DS shown on a large television screen can cause extremely large disparities. Our method is able to adapt content automatically to different output screens, given a pre-defined comfortable target depth range for an average viewer.

6.3 User Study

In order to further evaluate the utility of our method we conducted a user study with 31 subjects. The study was aimed at comparing our stereoscopic controller to standard stereoscopic rendering with fixed camera convergence and interaxial separation. All subjects were tested for proper stereoscopic vision using a random dot stereogram test. The goals of this study were

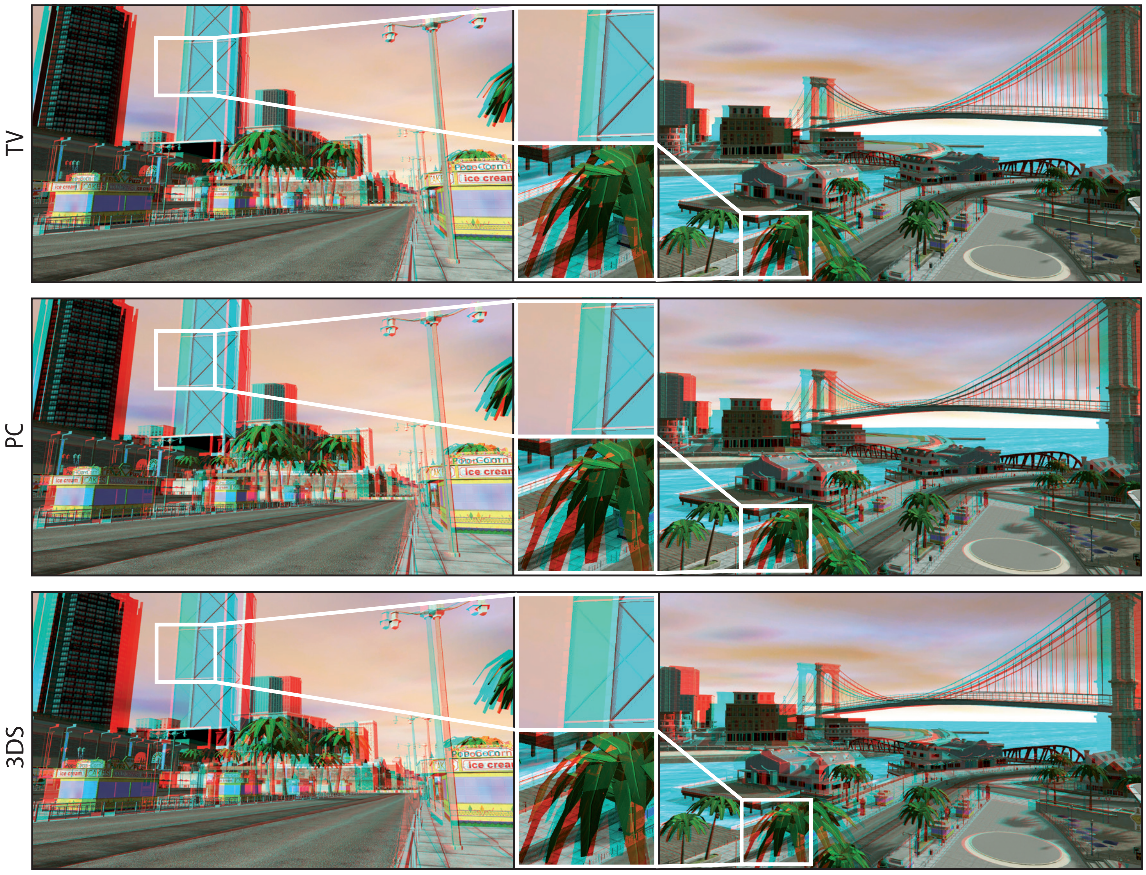


Figure 19: *Content produced for different screen sizes. The viewing conditions and target depth ranges for each device can be found in Table 2. In the middle of each row, magnifications of certain parts of the vistas are shown. The differing viewing conditions demand different disparities for the depth image to be perceived in the desired range.*

twofold. First, we tested whether the subjects prefer the static, uncontrolled stereoscopy or our constrained approach as more comfortable. Second, we examined if our controller compromises perceived realism due to the involved adaptation of camera convergence and interaxial separation.

To this end we rendered 10 side-by-side comparisons of different scenes using uncontrolled stereoscopic parameters and using our controller for pairwise evaluation [David, 1963]. Figure 14 shows two of the examples we used for this study. The rendered scenes contained typical scenarios encountered in interactive 3D environments, including

- continuous view changes between close-ups and wider scenes
- objects suddenly entering the field of view
- three long sequences where a player explores a complex game environment

We randomized the order of scenes as well as the respective positions on the screen. Each pair was shown three times so that the viewers had sufficient time for comparison. The study was performed on a line-wise polarized 46inch Miracube display. The viewing distance was 3 m, and our controller was configured for a target depth range of [-51.4, 86.5] cm with respect to the display. The static stereoscopic parameters were set such that at the beginning of each scene the resulting disparities were identical to our controller. According to our above mentioned goals, for every comparison the participants had to answer either *left* or *right* for the following two questions:

Q1: Which one is more comfortable to watch?
Q2: Which one looks more realistic to you?

When considering all 10 scenes in the evaluation, we received 310 votes for each of the two questions. Regarding question 1 about comfortable stereo viewing, our controller was preferred in 61.7% (191 of 310) of the examples, while the fixed stereo was preferred in 38.3% of the cases. In terms of realism, the results of our controller were preferred in 60.7% (188 of 310) of the scenes compared to 39.3% for the static stereo settings.

One stereoscopic issue that has not been considered by the stereo controller proposed in this thesis is the problem of so-called frame violations: if an object with negative disparity, i.e., in front of the screen, is cropped at the screen borders, the human visual system can get confused. This can be uncomfortable to the viewer. For the results used in this study, our stereo controller mapped the complete scene into a target volume [-51.4, 86.5] cm around the display. This introduced frame violations in some situations. We deliberately did not correct for such frame violations in order to evaluate the effects of depth remapping only. However, such a correction is trivial to add by adding corresponding *floating windows* [Gateau and Neuman, 2010; Smolic et al., 2011]. Therefore, if we remove the two sequences from the evaluation where the most obvious frame violations occurred (resulting in 248 answer per question), the preference for our method in terms of comfort raises to 70.9% (176 of 248), and in terms of realism to 69.3% (172 of 248). All these results are statistically significant with a p-value < 0.01.

From these results we can conclude that the stereoscopic imagery optimized by our controller was generally preferred by the subjects and created a more comfortable viewing experience without compromising perceived realism of scene depth. In addition, the results indicate an interesting correlation between comfort and perceived realism that we did not anticipate. In 86.1% of the answers the more comfortable rendering was also selected as the more realistic one. This is interesting since the dynamic adaptation of baseline and convergence and the resulting scaling of perceived depth over time seems to be less compromising in terms of perceived realism than excessive disparities.

7 Summary and Limitations

The stereoscopic camera controller that we have presented in this chapter has been developed to provide a practical, fast, and fail-safe solution for stereoscopy in high-performance interactive applications such as computer games. In order to avoid excessive disparities, we are constraining the camera separation and convergence distance to exactly map defined points in the virtual environment around defined distances in real space. Furthermore, we have identified the problem of parameter interpolation over time for changing scene contend and proposed a solution that allows the perceived space to transform according to interpolation functions.

7.1 Limitations and Future Work

The disparity optimization framework only manipulates the two most basic stereoscopic parameters, the camera convergence and interaxial separation. This allows for an analytical solution that is very fast to compute, but it is only a solution in a two-dimensional configuration space. While this is the most practical solution for real-time environments, we would like to investigate techniques for more complex nonlinear disparity remapping. Our experimental study provides encouraging evidence that this might be even more beneficial for the viewer. However, our study is only a first indicator that adaptive stereoscopy can increase viewer comfort. Additional studies need to be conducted to better understand the effects of such a stereoscopic control.

Furthermore, our method is designed for interactive environments without control over the camera movement. However, as we only manipulate the camera separation and convergence, nothing would prevent our method from working with real cameras, too. We would like to investigate the possibility to implement our method on a stereoscopic camera rig such as the one by Heinzle et al. [2011].

The linearized temporal interpolation of the stereoscopic parameters intuitively seems to work well for adjusting stereoscopy on-the-fly. However, it is not clear yet if the linearized interpolation is optimal w.r.t. the viewers comfort and impression of realism. We would like to further investigate the effect of our linearized interpolation on the viewer's perception.

Finally, he geometrical setup we have used for the derivation of our constraints is based on one viewer sitting centered in front of the screen. While this model holds up very well for a standard user-screen setting, the extension of our approach to multiple viewers is non-trivial. A horizontal shift of the viewer relative to the screen causes the reconstructed depth of the stereo footage to shear [Held and Banks, 2008]. This can be countered by adding a specific value to the image-shift of the cameras rendering the scene. Given a group of viewers, distributed over space in front of the screen. One could formulate Kalman- or Markov filters to estimate an optimal setting for the camera parameters.

CHAPTER 5

Fast and Flexible Color Balancing Using Example Images

Modern cameras are able to process photographs while they are captured. Many adjustments are made to enhance the image content in order to make life easier for the photographer. Unfortunately, this processing changes the image appearance in ways that is hard to reproduce. Using our color balancing approach, such color transfers can be easily extracted and applied to any other input image in order to enhance the realism of virtual footage.

Pablo Picasso highlighted the importance of color in painting with the famous quote, *"They'll sell you thousands of greens. Veronese green and emerald green and cadmium green and any sort of green you like; but that particular green, never."* Today, the same struggle to capture the perfect nuance of color lives on in digital photograph and video recording. The colors that are ultimately recorded by both consumer-grade and high-end digital cameras depend on a plethora of factors and are influenced by complex hardware and software processing algorithms, making the precise color quality of captured images difficult to predict and control. As a result, photographers and videographers must rely on post-processing algorithms to fix color problems. In turn, virtual cameras, that are not implementing such characteristic behavior, can seem dull and artificial.

This problem is generally known as color balancing or color grading, and plays a central role in all areas involving capturing, processing and displaying image data. In digital photography and filmmaking, specifically trained artists work hard to achieve consistent colors for images captured with different cameras, often under varying conditions and even for different scenes. With today's tools this process requires considerable, cost-intensive manual efforts. Similar issues arise in augmented reality applications, where computer generated graphics must be embedded seamlessly and in real-time into a video stream. Here, achieving consistent colors is critical but highly challenging, since white balance, shutter time, gamma correction, and other factors may continually change during acquisition and cannot be perfectly controlled or addressed through calibration alone. Therefore, an efficient correction and adaptation of colors that is agnostic to the underlying capture hardware is highly desirable in such AR-related applications. However, aside for a few preliminary efforts [Klein and Murray, 2010; Knecht et al., 2011] practically applicable solutions to this problem are yet to be developed.

Several research areas focus on ways to change the colors of digital images for different purposes. Automatic color balancing algorithms estimate the scene illuminant. More sophisticated approaches transfer color distribution statistics, such as color histograms, between images. These methods can produce appealing results, but require dense correspondences between image pairs. This limitation makes it difficult to apply existing color balancing or color transfer algorithms in real-time applications where the input images are continually changing.

Our work targets this limitation by focusing on real-time color balancing using sparse correspondences. This allows us to change colors in images based on example photographs taken with a digital camera. This extends the range of effects that are possible to perform in virtual cameras and allows the reproduction of characteristic camera color transfer effects such as adjustment of gain, contrast, etc. Furthermore, we show how to extend our approach robustly to the temporal domain, which allows color grading of rendered footage to real video. This extension allows more realistic rendering in applications such as augmented reality.

The remainder of this chapter is organized as follows. First, in Section 1, we discuss the current state of the art in color transfer and balancing. Then, in Section 2, we introduce our vector space color transfer using normalized radial basis functions. In Section 3, we show how basis functions can be trained on example images to achieve globally optimal balancing of unreferenced colors. In Section 3, we show a fast GPU implementation of our approach before we demonstrate its wide range of applicability in Section 4. Finally, we extend the color balancing approach to the temporal domain in Section 5 to enable more realistic augmentation of images and video.

1 Background of Color Balancing and Transfer

In this section, we discuss related work on histogram matching, user controlled color transfer, color calibration, colorization of greyscale imagery, and color balancing for image augmentation.

Histogram matching: This is the process of transferring the color distribution statistics from one image to another image, pioneered by Reinhard et al. [2001]. They proposed computing the mean and standard deviation of the color distribution in the color space. Then, by scaling and translating the parameters onto the ones of the target image, they achieve automatic color transfer. Xiao et al. [2006] extend this method to transfer per-channel color statistics in any color space and Pitié et al. [2007] introduce a minimum cost linear mapping using the Mongue-Kantorovitch theory of mass transportation. Prikli et al. [2003] perform image color histogram matching using dynamic programming. This way, they can automatically find a non-linear transfer from one histogram function to another one. Further extensions have been proposed that optimize the images after matching to avoid bad color gradients [Neumann and Neumann, 2005; Xiao and Ma, 2009]. Kagarlitsky et al. [2009] utilize histogram matching between different regions of a pair of images to achieve piecewise consistent color mapping.

The work of Chang et al. [2007] on example-based color transfer employs matching between color categories. All colors in an image are assigned to one of 11 basic color terms used in modern languages (e.g. red, green, blue, yellow, brown, etc.). Using this labeling, they realize an automatic mapping of the color between images of different content.

While histogram matching methods for automatic color transfer can produce very appealing results, they provide only limited control. Exact color matches are often required in image editing and are difficult to achieve with these methods. Furthermore, these methods are not suited for augmented-reality applications where the color distribution in the video can change considerably.

Interactive Color Transfer: Abadpour et al. [2004] first proposed an image color transfer algorithm with user interaction. In their work, the user is allowed to denote corresponding regions between images. Colors are then matched using fuzzy principal component analysis. Other methods have been proposed that aid the user in local color matching [Tai et al., 2005; Lischinski et al., 2006; Maslennikova and Vezhnevets, 2007]. The user is allowed to select a local region, and the actions are propagated throughout the rest of the image. However, due to the nature of manipulation, it can be difficult to match an exact color, even if the color is available in a target image. Also, histogram matching has been extended to allow user interaction [Wen et al., 2008; Pouli and Reinhard, 2011]. However, these methods are tailored to work between two images, and cannot be extended easily to video data, and therefore lack robust tracking of colors in video images.

The work of Levin et al. [2004] on colorization of greyscale video sequences propagates user scribbled colors. Their approach is based on the premise that similar intensity in neighboring pixels in space-time has similar color. An et al. [2010] also propose a scribble-based framework for color transfer between images. For each pair of strokes, a non-linear color transfer function is estimated. In a second step, the estimates are refined using a global optimization. Our work, in contrast, targets color balancing where only a sparse and possibly incomplete set of correspondences are available.

Fast and interactive methods for editing images and video have also been proposed. An et al. [2008] propagate local edits by assisting a user with iteratively refining rough edits to areas of spatial and appearance similarity. Farbman et al. [2010] propose the use of diffusion maps to propagate image editing information. Both approaches are robust to highly textured image data and work at interactive speeds. However, they lack the performance needed for demanding real-time applications.

Li et al. [2010] provide an approach for image and video editing with instant results. They employ radial basis function interpolation in a feature space to propagate sparse edits locally. In contrast to this work, we are formulating the problem as a global image space transformation. Furthermore, we propose an approach to learn the shape of the basis functions using example images and show that the Gaussian function used in their method is sub-optimal for global color balancing.

Color Calibration and Balancing: Color calibration is the process of computing the color distribution function of a camera to allow recorded images to be consistently colored. Adams et al. [] gives an overview of standard camera calibration techniques using linear transformations. Usually, a 3x3 matrix is computed using regression performed on a set of colors with known values. Image grading, on the other hand, is concerned with equalizing the color distribution of video footage. Pitie et al. [2007] propose an automated iterative process to morph the color palette of one image into that of the other image. The work of Senanayake et al. [2007] solves a similar problem using histogram matching with the concept of feature registration. Their approach targets equalizing images of the same scene where color histograms show very similar peaks and features. The main difference between grading algorithms and our approach is that we are not relying on a dense set of color correspondences.

A related area is color consistency. Humans perceive colors to be consistent even under different lighting conditions. This property is not available for digital images. Colors will have different numerical values under different illuminations. In order to counter this issue, different algorithms have been proposed that try to estimate the scene illuminant in order to color balance the images. An overview over different algorithms is provided by Argawal et al. [2006] and Cohen [2011].

In contrast to our approach, many of these methods do not apply additional information such as correspondences or known colors in a scene to balance the images. Therefore, these methods are less interactive and less robust when applied to videos. The work of Farbman et al. [2011] on color balancing approaches the problem of tonal inconsistency in videos. They propose using anchor frames. By propagating per-frame adjustment maps in color space, they achieve a smooth stabilization of the video colors. However, the work relies on dense tracking of the image content.

There is also significant work that enhances the quality of images using a database of example images [Siddiqui and Bouman, 2008; Dale et al., 2009; Wang et al., 2010; Kang et al., 2010]. The work of Yang et al. [2011] achieve color compensation by replicating the non-linear relationship between the camera color and luminance. Wang et al. [2011] transfer color and tone style of expert photographed images onto the images from cheap cameras by learning the complex transformation encoded in local descriptors. Our parameter optimization of the radial basis functions is also based on extracting image properties form data sets. However, we design our functions to be sparsely sampled so that they provide good results when interpolated in order to considerably increase performance.

Another field where images are color balanced is panorama stitching. Here, however, there is only partial overlap of the images and therefore different approaches have been proposed, like non-linear regression [Pham and Pringle, 1995], histogram map based color correction [Tian et al., 2002], and light-weight color and luminance compensation [Xiong and Pulli, 2010].

Colorization of Greyscale Images: A related field of research is the colorization of grayscale images. Welsh et al. [2002] colorize grayscale images by transferring chroma values from a target image using user selected rectangular correspondence areas. Ye et al. [2004] transfer color to greyscale images by analyzing the texture around each pixel. Recent work by Sang et al. [2011] employs segmentation and the use of reference images from the internet to create robust colorization. Hertzmann et al. [2001] employs an image-analogy framework that supports colorization. These methods solve a similar problem to ours. However, they involve per-pixel neighborhood analysis, segmentation, or a best-match search. These operations are time consuming and preclude interactive use.

Color Balancing for Image Augmentation: Only recently there has been research in improving the compositing of images for augmentation and augmented reality. Klein and Murray [2010] examine and reproduce several image degradation effects of low-cost digital cameras to increase the realism of augmentation. While their work includes a vast number of effects such as lens distortion, blur, bayer artifacts, and more, they are not considering color shifts due to camera parameter changes. The work of Knecht et al. [2011] proposes to balance rendered images using colors in the scene. They are then applying a linear regression model to adapt rendered footage. While their approach is able to adjust the color balance to video footage, their color balancing is limited due to its linearity.

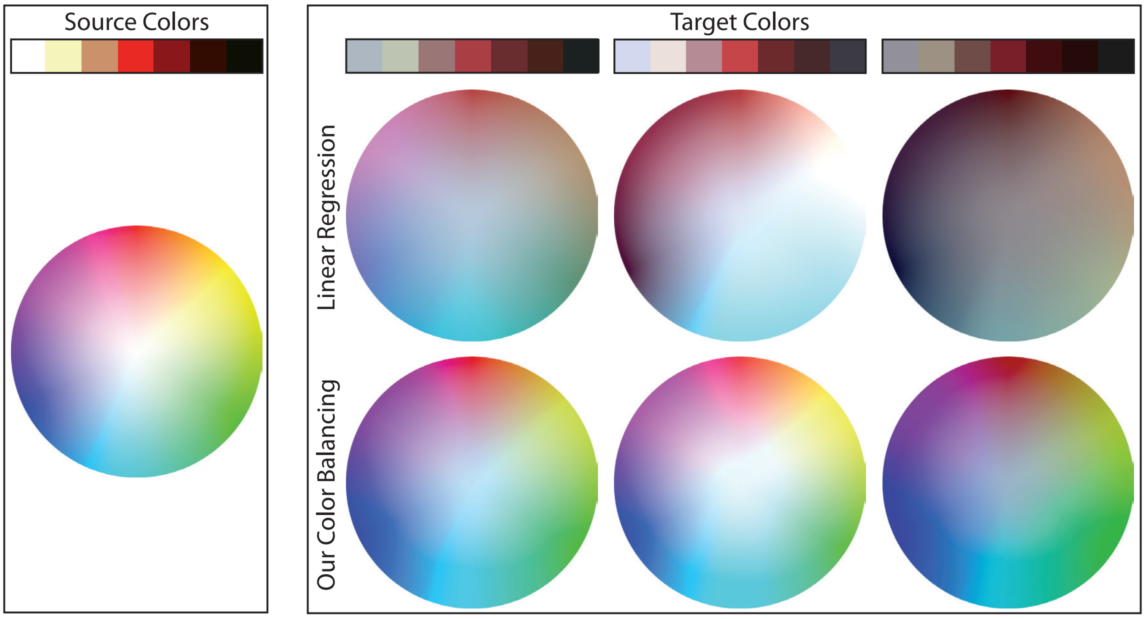


Figure 1: *Color balancing when only a limited number of color references are available. On the left, the set of source colors (dominantly from the red area) are shown together with the original color circle. On the right, a comparison between standard linear regression color balancing and our method is shown. It can be observed that our method balances colors that are not referenced (blue and green) better than the standard approach.*

2 Vector Space Color Balancing

Given a sparse set of color correspondences, the goal is to transform the color gamut of an image into a different shape such, that (i) colors that have references are transformed as close as possible to that point in the color space, and (ii) colors, for which no references are available, are transformed in a plausible way.

A standard way to perform such a color space transformation is to use an affine transformation found by linear regression on the correspondences [Adams et al., ; Knecht et al., 2011]. This transformation can be computed analytically and produces plausible results for colors inside the convex hull of the sample colors. This approach works best if there is a wide range of colors available whose convex hull covers as much as possible of the color space. Usually, color checker are used in such a scenario, as they provide a good range of colors, often times optimized for certain tasks, like capturing human skin tones in video recording.

However, there are several drawbacks to the standard calibration with linear

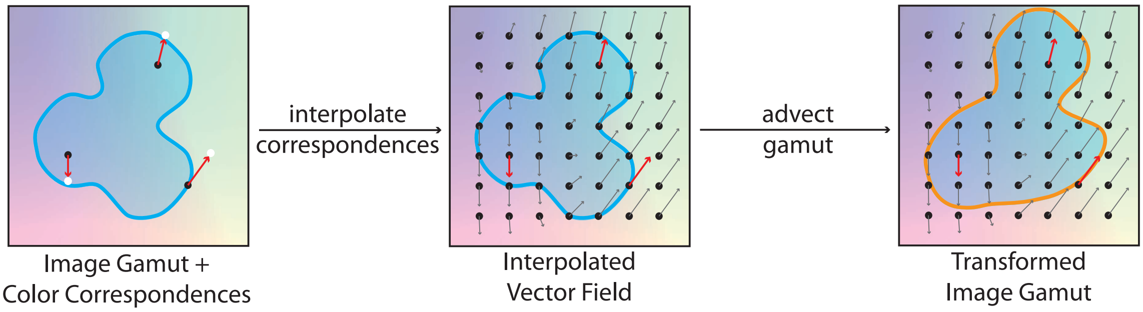

interpolate correspondences
advect gamut
Image Gamut + Color Correspondences
Interpolated Vector Field
Transformed Image Gamut

Figure 2: *The idea of vector field color transfer. On the left, the source colors are shown in black and the corresponding target colors in white. The idea is to interpret each correspondence as a vector in color space and to extend them to a vector field with which the gamut of the image is transformed. This procedure creates a continuous and flexible color transfer function.*

regression that can cause artifacts in image processing. First, the affine transformation is not always able to transfer all colors correctly. The regression approach to find the transformation linearizes the transformation over all points. If a subset of the color correspondences needs significantly different transformations as the rest, their constrained colors will not be reached completely. More complex regression methods that employ higher order terms exist [Adams et al.,]. In extreme cases, however, also a higher order regression might not be enough. Second, affine transformation can fail outside the convex hull. This is because of the linear nature of the approach. Noise present in the given color correspondences can be amplified outside of the convex hull, and can create unnatural color changes. This is especially apparent when only a small sub-space of the color space is covered by the convex hull. Figure 1 shows some examples of linear regression transformation on a color circle. Source colors, mostly in different red tones, are transferred to three different sets of target colors. It can be observed, that linear regression balancing causes artifacts in the colors not present in the color references.

Knecht et al. [2011] propose a solution to the second problem. Their approach assumes a dominant color, for which correspondences are available. A dominant color is identified as covering a large range in one of the three main color axes (red, green, or blue). The transfer profile shown by this dominant color is then applied to the other color space dimensions. This approach basically mirrors the color transfer function to the other axes in the color space. In contrast to this approach, we want the color transfer function to automatically extrapolate the behavior of the reference colors outside its convex hull.

We solve the problem of color balancing by interpreting the given color correspondences as vectors in the color space. In order to produce a transformation of the gamut, we compute a smooth vector field around the reference vectors that achieves the aforementioned goals. The concept is demonstrated in Figure 2. The correspondence vectors themselves represent constraints in the color space. In a first step, we propagate the constraint vectors through the CIE La^*b^* space using scattered data interpolation to create a smooth vector field. Then, in a second step, the image gamut is transformed using this vector field to produce a color balanced image.

In order to be able to use the length of the vectors and other distance measures in a linear fashion, we perform all color related operations in the CIE La^*b^* space with D65 standard illuminant. This space is designed such that distances are measurements for perceived color difference. There exist several distance measures, namely CIE DE76, DE94, and DE2000. Generally, we use Euclidean distance (DE76) as it is the easiest to implement and the most efficient to compute. However, the other distance measures can be applied without loss of generality.

2.1 Vector Field Computation

Given a list of source points $(c_1, c_2, \ldots, c_n)$ in the three-dimensional color space, and a list of target points $(d_1, d_2, \ldots, d_n)$ in the color space to which the source colors should be mapped. The goal is to find a color transfer function R that transforms all colors in the three-dimensional color space such, that the c_i are mapped as close as possible to the d_i. This leads to the constraints

$$d_i = c_i + \mathbf{v}, \tag{1}$$

where the $\mathbf{v}_i$ are the constraining vectors at the color points c_i. For all unreferenced colors, these constraint vectors $\mathbf{v}_i$ should be propagated smoothly into the rest of the color space so that the remaining colors are transformed along with the constrained colors, as demonstrated in Figure 2. Since we assume a sparse set of colors c_i that are provided with corresponding target colors d_i, the vector field can be computed through scattered data interpolation.

Normalized Radial Basis Function Interpolation

The problem at hand, namely to create a smooth vector field given individual vectors in three dimensional space without explicit connectivity, is known in the literature as scattered data interpolation. Generally, scattered data interpolation is employed to create a smooth surface given a (possibly sparse) number of points in space. This concept can be easily extended to volumes.

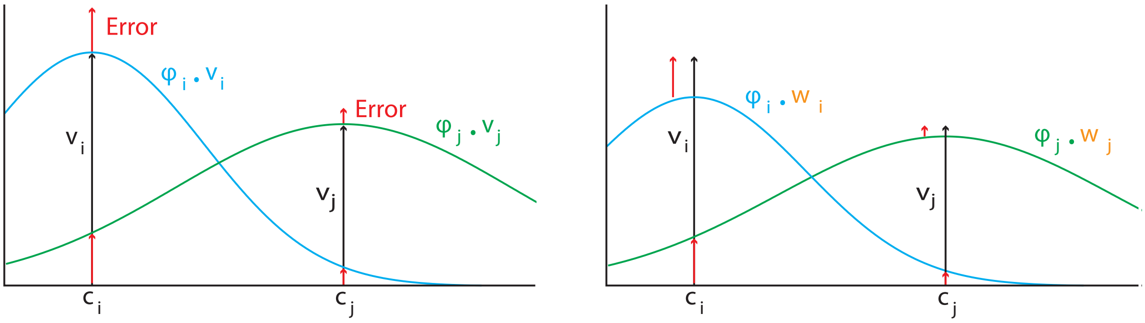


Figure 3: *Visualization of the problem of overlapping support between interpolation functions. When summing up the support of each function at the data points c_i, even a small overlapping support from the other data points causes an error. RBF interpolation optimizes a vector $\mathbf{w}_i$ for each interpolation function such, that the sum of all overlapping support gets as close to the supposed data value as possible.*

Scattered data interpolation in our setting attempts to compute, for each new color point e in the source space a vector $\mathbf{v}_{\text{new}}$ based on the surrounding $\mathbf{v}_i$.

Radial basis function (RBF) interpolation is a way to define the spread of data through a function depending on increasing distance. For a data vector $\mathbf{v}_i$ of each source color c_i a function ϕ_i is defined that describes the spread of the data vector in space. By normalizing and summing up these functions at a new color e, the $\mathbf{v}_i$ can be smoothly interpolated to

$$\mathbf{v}(e) = \frac{1}{\sum_{j=1}^{n} \phi_j(c_i)} \sum_{j=1}^{n} \phi_j(e)\, \mathbf{v}_j. \tag{2}$$

The problem with this kind of interpolation is, that, while $\mathbf{v}_i$ gets perfectly interpolated at c_i when only considering ϕ_i, the other functions falsify the result due to the overlapping support. Figure 3 visualizes this problem. When the reach of two functions overlaps, the interpolation at the data point itself receives an error. For functions like gauss, this cross-talk may be small, however, it can add up when the c_j get close enough. Normalized RBF interpolation tackles this overlapping support problem by defining weight vectors $\mathbf{w}_j$ for each ϕ_j such, that the data vectors $\mathbf{v}_i$ are constrained at c_i.

$$\mathbf{v}_i = \frac{1}{\sum_{j=1}^{n} \phi_j(c_i)} \sum_{j=1}^{n} \phi_j(c_i)\, \mathbf{w}_j \tag{3}$$

This means, that for each function ϕ_i a weight vector $\mathbf{w}_i$ should be found such, that when all functions are summed up and normalized, they provide a value as close as possible to $\mathbf{v}_i$ in the least squares sense.

$$\arg\max_{\mathbf{w}}(||\mathbf{v}_i - \mathbf{v}(c_i)||^2) \tag{4}$$

The normalization makes sure that the $\mathbf{v}(e)$ interpolated at a color value e in between the source color points c_i gives a correctly scaled vector. The minimization in Equation 4 an be achieved by creating a system of equations. Per dimension, each component of the individual $\mathbf{v}_i$ and $\mathbf{w}_i$ are assembled into a large vector $\mathbf{v}^D$ and $\mathbf{w}^D$ $(D = \{L,a,b\})$. The problem can now be written as a system of linear equations for each of the three dimensions in the La^*b^*-color space

$$\begin{aligned}\mathbf{v}^L &= \mathbf{\Phi}\,\mathbf{w}^L\\ \mathbf{v}^a &= \mathbf{\Phi}\,\mathbf{w}^a\\ \mathbf{v}^b &= \mathbf{\Phi}\,\mathbf{w}^b.\end{aligned} \tag{5}$$

The matrix $\mathbf{\Phi}$ contains, at each position (i,j), the overlapping support value of c_j seen at the interpolation point c_i, normalized to the sum of the entire support.

$$\mathbf{\Phi}(i,j) = \frac{1}{\sum_{k=1}^{n}\phi_k(c_i)}\phi_j(c_i) \tag{6}$$

Since all three dimensions of the color space use the same matrix Φ, it only needs to be inverted once to solve all three equation systems

$$\begin{aligned}\mathbf{w}^L &= \mathbf{\Phi}^{-1}\,\mathbf{v}^L\\ \mathbf{w}^a &= \mathbf{\Phi}^{-1}\,\mathbf{v}^a\\ \mathbf{w}^b &= \mathbf{\Phi}^{-1}\,\mathbf{v}^b.\end{aligned} \tag{7}$$

Finally, this gives the weight vectors $\mathbf{w}_i = (\mathbf{w}_i^L, \mathbf{w}_i^a, \mathbf{w}_i^b)^T$ that can be used for interpolating the color transfer of the entire color space. The transfer vector $\mathbf{v}(e)$ for an arbitrary color e can now be computed as

$$\mathbf{v}(e) = \frac{1}{\sum_{i=1}^{n} \phi_i(e)} \sum_{i=1}^{n} \phi_i(e)\, \mathbf{w}_i. \tag{8}$$

The advantage of normalized RBF interpolation is that the interpolation functions can be freely chosen, and still are able to achieve color references as close as possible. Additionally, the evaluation of the functions at each point in the color space supports a parallel implementation and thus a fast solution. The solving of the equation is a drawback of the method as it involves a matrix inversion. This inversion is run-time dependent on the number of references, however, the inversion can be performed in real time for more than 100 elements, and only needs to be calculated once for a fixed set of referees.

Discussion For the purpose of creating a smooth vector field based on a sparse set of correspondences, RBF interpolation provides a good balance between complexity of computation and flexibility of the distribution of data that suits our problem. The main advantages leading to this choice are the least squares approximation of the constraints while still allowing the basis functions to be arbitrary shaped. In Section 2.2 we use this to our advantage by optimizing the shape of different basis function types to get the best reconstruction of colors in a set of example images.

A problem with RBF interpolation are conflicting constraints of two source colors. If two colors c_i and c_j with a small distance in the color space have opposing data vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, the matrix $\mathbf{\Phi}$ can become near-singular. This causes components of the resulting $\mathbf{w}_i$ to become negative. Negative values create an inverted distribution of the constraint vectors $\mathbf{v}_i$ in the color space and produce undesired artifacts.

This problem, in our specific setting, can be easily circumvented. Since we are dealing with color values in La^*b^*-space, we have a notion of distance. Thi La^*b^*-space is specifically designed to be linear in the perception of color distances, and scaled such, that a distance of one is defined as the smallest noticeable color difference. Even a distance of two is still considered barely noticeable under perfect viewing conditions [Wyszecki and Stiles, 1967]. This property allows us to perform a neighborhood clustering of source colors c_i before the computation of the weights. By combining all the references that lie within a radius of one in the color space, we make sure that the color references still remain perceptually the same. In our implementation we performed a greedy clustering approach, where we select a source color at random, and then cluster all colors that lie within one unit around this color in the La^*b^*-space. The reference color representing this cluster is computed as average of all the source colors, and the transfer vector is computed as the average of the according vectors as well.

The clustering approach additionally provides another beneficial property to our color balancing approach. For data sets where there is a lot of redundancy in the references, the clustering can be used to perform a natural reduction of the number of references, as well as a smoothing of the noise contained within these references. The clustering radius can be regarded as a valuable parameter for performance crucial tasks, where an increase of the radius reduces the number of correspondences. Since the clustering is performed in the La^*b^*-space, the reduction of references is perceptually motivated.

In addition to clustering conflicting constraints, the choice of the basis function also influences the stability of the matrix $\mathbf{\Phi}$. Functions with a wide spread in the color space can cause $\mathbf{\Phi}$ to degenerate quicker. In Section 2.2 we are comparing the performance of different functions in their performance of reconstructing unknown colors. We show that an adapted version of the Shepard function is able to achieve good reconstruction results with a very small spread, and thus provides an excellent choice of basis function.

2.2 Selecting and Optimizing Interpolation Functions

With the approach of radial basis functions, we can make sure that overlapping support between basis functions is minimized. However, it is still unclear what functions to consider for interpolation and what parameters to use. We therefore compare the performance of different functions on reconstructing colors for which no references are available. The goal is to find an optimized function that is able to recreate the non-linear transfer function of a real camera. For this purpose we create several data sets by filming a color-checker and changing camera parameters like e.g. exposure time or gain. This way, we can capture the color transfer functions of the individual effects. In each of these sets we randomly select color patches that are used as references. The performance of the functions is then tested by comparing how close they can reconstruct the missing colors.

In our experiments we consider the following three functions that show different fall-off behavior. Each function is parametrized with an ϵ-value that allows to change its shape. This parameter will be subject to optimization.

Adapted Shepard: This function is an adapted version of the original Shepard interpolation function [Shepard, 1968]. We have changed the function such that it peaks at 1 instead of infinity (by adding $+1$ to the basis of the function).

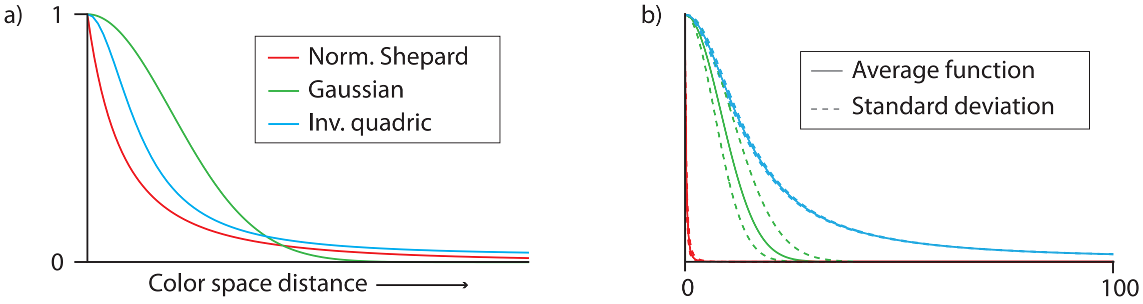


Figure 4: *The radial basis functions used to compute the color transfer vector field are shown. a) The three functions are shown that we have tested in our experiments. The functions are plotted with different ϵ values to show their fall-off behavior in the same scale. (b) The three radial basis functions are shown after optimizing the ϵ parameter with the dashed lines indicating the standard deviation found in the experiments. It can be verified that the Shepard function shows the quickest fall-off while the inverse quadric function has the widest spread. The Gaussian function showed the largest deviation during our tests.*

This makes the function more predictable and also numerically simpler to handle, as the special case of $||c_i - e|| = 0$ does not need special treatment.

$$s_i(e) = (1 + ||c_i - e||)^{-\epsilon} \tag{9}$$

Gaussian Function. In contrast to the Shepard function, the Gaussian function starts with a smooth falloff before declining exponentially. The Gaussian function also reaches smaller values much faster after a while than the Shepard function. This function with $\epsilon = 2$ can be considered a standard function for use in RBF interpolation [Li et al., 2010]. However, our goal is to investigate different ϵ values as well.

$$g_i(e) = e^{\epsilon ||c_i - e||} \tag{10}$$

Inverse Quadric Function. Finally, we also test the performance of a slightly adapted version of the inverse quadric function. We add a constant weight α to the function in order to guarantee a minimum value that the function assumes, even for distances at infinity. This addition allows, in contrast to the adapted Shepard and Gaussian functions, a more predictable interpolation of colors far away from the source points. In all our experiments we use a value of $\alpha = 0.01$.

$$q_i(e) = \frac{1 - \alpha}{1 + (\epsilon ||c_i - e||)^2} + \alpha \tag{11}$$

While testing the performance of each of the three candidate functions discussed above, we also want to find out the optimal ϵ value at the same time.

Thus, we perform the tests with changing ϵ values for each function and compare their capability of reconstructing color changes with the standard regression.

Evaluation

The evaluation procedure for one function is performed as follows. First, test data sets are created. For this purpose video footage is recorded of a color checker pattern (Digital ColorChecker SG in our experiments) in neutral illumination. During video capture, internal camera parameters (such as exposure, gain, or contrast) are changed individually so that the color changes are part of the data set. Then, in each data set, a frame, which is recorded with default camera settings, is used as base frame. In this frame, 50% of the color fields are selected at random, and used as interpolation points. Then, for one of the other frames in the data set (that contains a certain degree of distortion), the corresponding color fields are used to define color references. Now, in order to measure the performance of an interpolation function, the defined color references for the distorted frame of the data set are used to transform the remaining 50% of the color fields. The distance of these transformed colors to their ground truth value is then averaged to calculate the frame score. Then, for all frames in the data set, the frame score is computed, selecting the 50% interpolation colors randomly for each frame. The final score is obtained by averaging all frame scores.

This test is repeated for a range of ϵ-values for each function against data sets that are recorded under different camera distortion effects. The camera effects we have tested are shown in Figure 5; changing brightness (a), contrast (b), exposure (c), gain (d), color intensity (e), and white balance (f). Each test set contained between 150 and 350 frames, beginning with all the parameters set to default. In each set, the respective parameter was gradually changed to its maximum value, then its minimum value, and finally back to its original setting.

Test Results

The results in Figure 5 are plotted as average distances in the La^*b^*-space with the ϵ-values in the horizontal dimension. The score is shown as a curve for increasing ϵ-values for the adapted Shepard (red), the Gaussian (green), and the adapted inverse quadric (blue) functions in comparison to linear regression (black). For each function, the optimal ϵ with minimal error distance is marked with a triangle. There are a couple of interesting properties

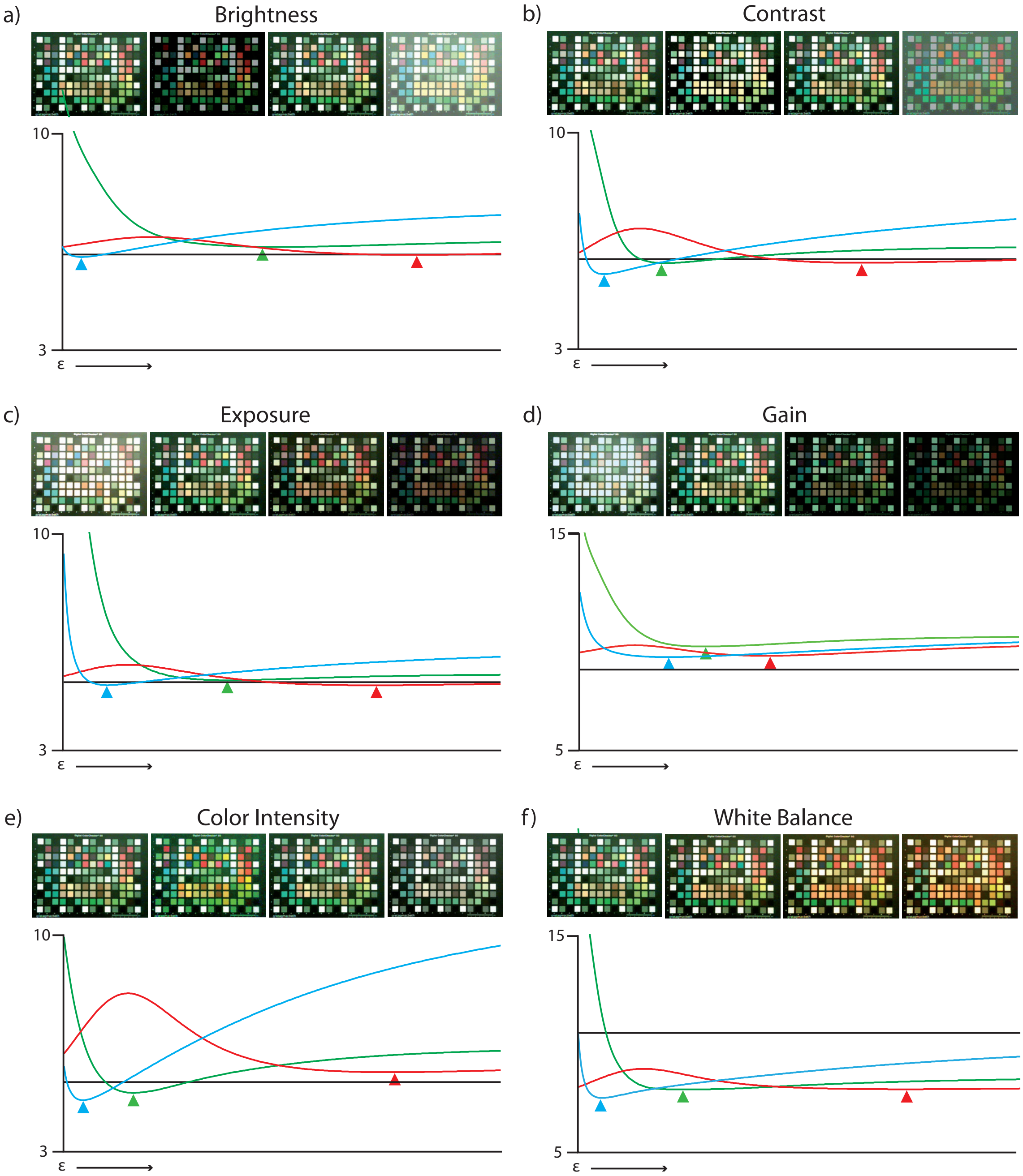


Figure 5: *Average reconstruction error of the considered basis functions while interpolating 50% of the color checker fields under changing camera parameter. The score is shown for increasing ϵ-values for the adapted Shepard (red), the Gaussian (green), and the inverse quadric (blue) functions in comparison to linear regression (black). The optimum of each function is marked with the colored triangle. Note that the functions are shown with differently scaled ϵ-values so that their optima are shown in a similar range. The scaling for each of the three function types, however, is consistent across all six plots. In most of the test scenarios, the RBF interpolation achieves smaller errors in reconstructing colors without reference values compared to linear regression.*

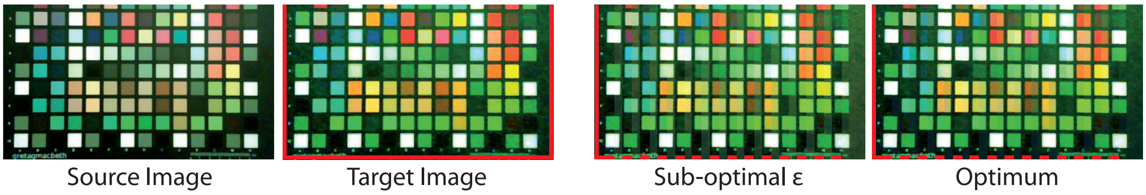


Figure 6: *Difference in color balancing when using different ϵ-values. The source pattern is balanced towards the target pattern using three color references. The examples show the target pattern (red border) interleaved with the balanced source pattern using the inverse quadric function with three different ϵ. As predicted by our tests, a poorly choosen ϵ-value (here, optimal value divided by 20) shows significantly more difference, while the balancing using the optimal ϵ is visibly closer.*

that can be observed in these plots, and provide a background for deciding which function is most appropriate to use for creating a vector field for color balancing.

The first property that can be observed in all the plots in Figure 5 is, that all the curves are smooth, even though we are selecting the reference colors at random for each frame of the data set. We have observed this behavior during all our test runs. This result lets us believe that the tests are robust enough to formulate statements about the performance of the functions.

Second, it can be verified that the RBF interpolation performs very similar to, and often better than linear regression in terms of reconstructing colors that have no reference. Note, that these graphs do not take the interpolation of referenced colors into account. Here, the RBF interpolation performs by definition optimal while the affine transformation shows errors in the same magnitude than with the interpolation of unknown colors (since it is a regression method). Therefore, the overall reconstruction performance of our approach exceeds the performance of the classical regression method.

The third interesting property of the plots is the fact that there is always a local optimum for a certain parameter ϵ. All three function that we have

	Shepard	Gaussian	inverse quadric
parameter value ϵ	3.7975	0.0846	0.0690
standard deviation σ	0.5912	0.0187	0.0289

Table 1: *This table shows the average ϵ-value and the standard deviation found determined by parameter tests.*

tested seem to have exactly one optimum that is non-zero. Only the adapted Shepard function converges to a second (local) minimum when ϵ converges towards zero. However, we ignore this case to avoid singularities during the RBF interpolation. Interestingly, for all the different color transfer effects we have tested, the local optima are around similar ϵ-values for each function. Note, that in the plots in Figure 5, the individual ϵ-values of each function are scaled so that they appear at a similar location in the horizontal axis. For each of the three types of basis functions, however, we have used the same scaling across all six plots.

The ϵ-values and their standard deviation for each function we have found are presented in Table 1. The functions using the optimal ϵ-values are plotted 4 b). The figure shows, that the inverse quadric function has the widest spread, while our adapted Shepard function has the smallest spread. The Gaussian function is in the middle, however, showing a significantly larger deviation than the other two functions. The fact that the Gaussian function shows the largest variation is interesting, as it is usually the standard function to be used in similar problems [Li et al., 2010].

Figure 6 shows a comparison between an optimized ϵ-value for color balancing and a poorly chosen one. On the left of the figure, the source and target images are shown. On the right, an overlay of the target and the balanced source is shown, where the red border indicates the ground truth (target). It can be observed, that a poorly chosen ϵ-value introduces a more noticeable difference in colors after balancing than when using the optimized value.

2.3 Choice of Basis Function

To this end, we have discussed an approach to optimize and compare different radial basis functions. We have tested three differently shaped functions and found that their performance in interpolating unreferenced colors is comparable. However, their shape and variance differs significantly, as shown in Figure 4 b). In the following we will discuss these differences and ague that our adapted version of the Shepard function is the best choice.

Ideally, the optimal function-shape should be fairly invariant to the image distortion effect in order to provide a wide variety of uses. Such invariance is indicated by a mall variation in a function's shape between different tests. As can be observed in Figure 4, the Gaussian function shows significantly more variation than our adapted Shepard and the inverse quadric functions. This indicates that the Gaussian function is a sub-optimal choice for global color balancing using normalized RBF interpolation.

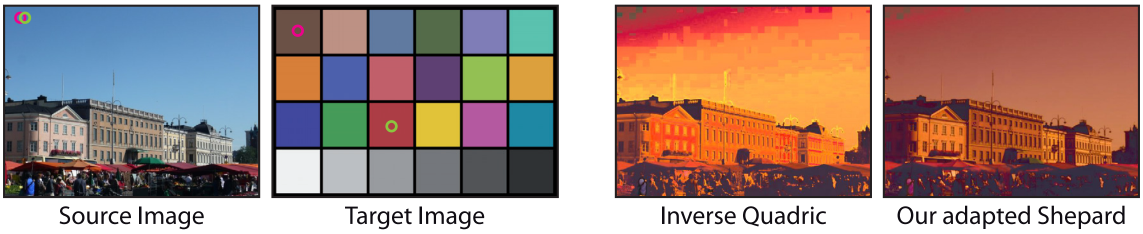


Figure 7: *The impact of the basis function's spread with close data points is shown. The source image is balanced using two different color references that lie very close to each other (indicated by the colored circles in the source image). On the right it can be observed that the inverse quadric function causes artifacts visible as false colors (e.g. the yellow lining along the roof). Our adapted Shepard function, in contrast, does not show these types of artifacts.*

Another important property of a function is its spread (i.e. the width of its fall-off curve). Radial basis function interpolation minimizes the overlapping support between two data points by solving a system of equations (see Section 2). The higher this overlapping support becomes, the closer the equation system comes to being singular. Even though we assure that the system is not singular by clustering color values that lie within the visible difference, the system can still become unstable when the overlapping support becomes to large. This results in an over-fitting of the constraints and can cause artifacts. Therefore, a function with a small spread is preferable to a function with a large spread. As seen in Figure 4 b), the adapted Shepard function has a very small spread compared to the inverse quadric function. Therefore, we propose to use our adapted Shepard function for the RBF propagation of the color transfer vector field. Figure 7 shows a comparison of interpolating close constraints using the adapted Shepard function and the inverse quadric function. The artifacts caused by the larger spread of the inverse quadric function are visible as false colors like the yellow lining around the roof or the red tint on the leftmost facade. The block artifacts, that are due to image compression errors, are also reduced significantly using the adapted Shepard function.

It is important to note that the way we are evaluating the performance tests is to find a generally applicable basis function. The data sets we used in our tests are probably biased towards the specific camera color transfer functions of the model we used. Nonetheless, we found that the functions performed very well in many applications. We show a series of examples using our adapted Shepard function with the optimal ϵ value (cf. Table 1) in Section 4. Figure 8 also shows the impact of the function in balancing an image with an increasing number of correspondences.

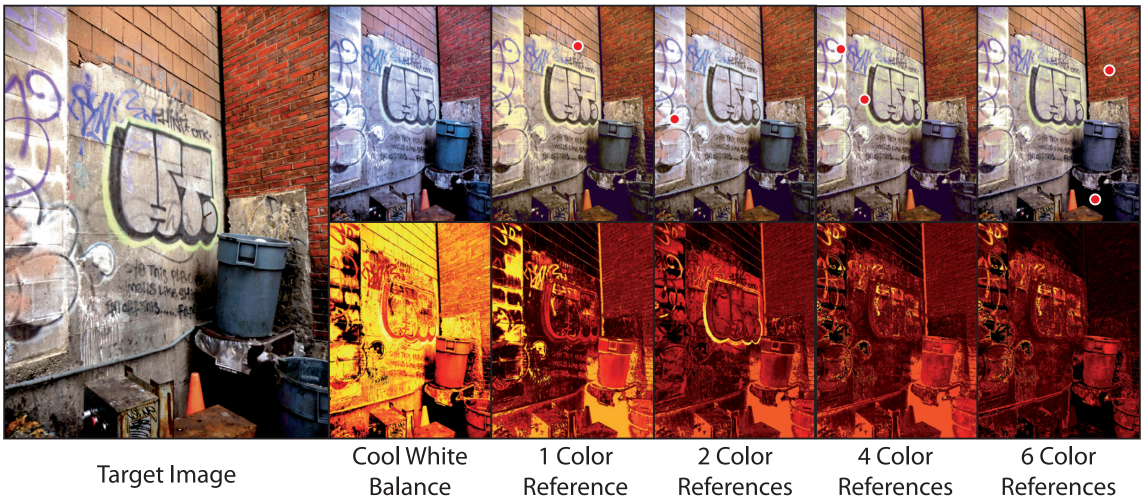


Figure 8: *Example of the global impact of our color balancing using only a few correspondences. The target image on the left is reconstructed using a photograph of the same scene with a different white balance as source. The top row shows the successive addition of references (red dots). The images in the bottom row visualize the difference in CIE La*b* space between the balanced image and the original.*

Our approach for selection and optimization of the functions naturally extends to different scenarios (different color checker, camera, and parameter) and other basis functions. In fact, our approach provides a tool to learn both specific camera characteristics as well as general camera behavior.

3 GPU Implementation

Our color balancing algorithm can be implemented by simply extending any given fragment shader. The reference colors c_i and weight vectors $\mathbf{w}_i$ can be transferred to the GPU as texture data. The matrix inversion is calculated beforehand on the CPU as it only needs to be performed once. The GPU fragment shader algorithm is described in Algorithm 2.

Algorithm 2: *Fragment Shader Color Balancing*

```
 1: $e \leftarrow$ pixelColor()
 2: $e_{Lab} \leftarrow$ rbg2lab($e$)
 3: $\mathbf{v}_e \leftarrow (0,0,0)$
 4: $s_e \leftarrow 0$
 5: for $i = 1 \rightarrow n$ do
 6:     $[c_i, \mathbf{w}_i] \leftarrow$ dataTexture($n/i$)
 7:     $r \leftarrow$ phi($c_i, e_{Lab}$)
 8:     $\mathbf{v}_e \leftarrow \mathbf{v}_e + r\,\mathbf{w}_i$
 9:     $s_e \leftarrow s_e + r$
10: end for
11: $e_{Lab} \leftarrow e_{Lab} + \mathbf{v}_e / s_e$
12: return lab2rgb($e_{Lab}$)
```

First, the original color e of the pixel is determined (line 1). The pixel color is the result of the given shading algorithm. This color is then transformed to CIE La^*b^* space (line 2). Then, the variables $\mathbf{v}_e$ and s_e are initialized (line 3 & 4). They are used to compute intermediate results of the transformation vector computation. Now, a loop is performed n-times, which is the number of color correspondences. In each loop, first, the color c_i and weight vector $\mathbf{w}_i$ are acquired through texture lookup (line 6). Then, the function ϕ is applied using the current color e_{Lab} and the support color c_i (line 7). This gives the function value r, which is used to add the current weight $\mathbf{w}_i$ to $\mathbf{v}_e$ (line 8). Additionally, the sum of all the function values r is stored in s_e (line 9), so that after the loop the transformation vector $\mathbf{v}_e$ can be normalized and used to translate e_{Lab} (line 11). The final color of the pixel is then transformed back into RGB space and returned to the frame buffer.

We have implemented the color balancing algorithm on an Intel Core i7 3.2 GHz with Nvidia GeForce GTX 460. The run-time in ms is shown in Figure 9 for an increasing number of references. The solver for the equation system (blue graph) is the bottleneck, as it involves a matrix inversion in the size of the number of references. The RBF interpolation (green graph)

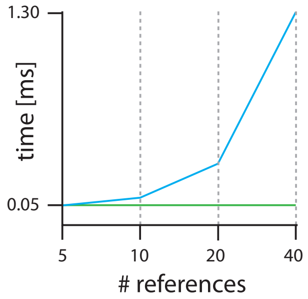
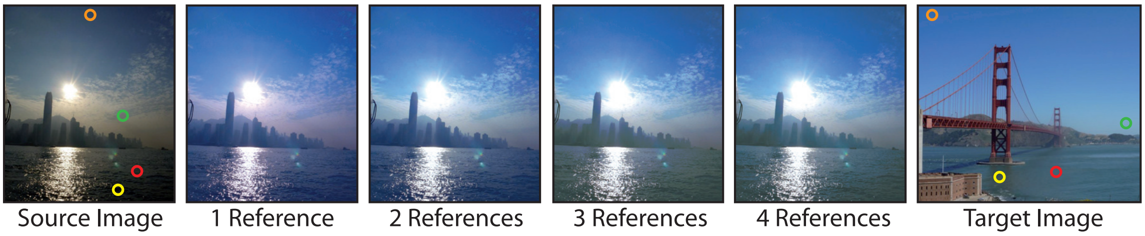


Figure 9: *The performance of our implementation. The green function shows the run-time for the color balancing on the GPU, the blue function the time to solve the equation system on the CPU.*

however performs with around 0.05 ms almost independently of the number of references.



Source Image 1 Reference 2 References 3 References 4 References Target Image

Figure 10: *An example of our color balancing an image using an image of similar content. With four correspondences the source image is balanced towards the target image. Using only two references for the sky (orange and green circles) and two references for the water (yellow and red circles), the source image's quality is improved significantly. Additionally, the sky and water in the balanced source image adopt the exact colors from the target image.*

4 Applications and Results

In the previous sections we have discussed an approach to color balancing of images given only a sparse set of correspondences. Through our optimization of basis functions we provide a tool to learn color changes and optimally adjust images accordingly. Additionally, we have shown a fast implementation that is able to perform color balancing with 40 references in less than 2 ms. In the following we discuss applications of our method for color balancing, image style transfer, and the reproduction of unknown color transfer functions from digital cameras using example photographs.

4.1 Interactive Color Balancing and Style Transfer

The fast implementation of our color balancing algorithm directly provides an easy-to-use tool for color balancing. The user is presented with two images, the one he wants to change, and the one from which he wants to reference colors. Color correspondences can now be defined by simply clicking on the images. The result is instantly visible, and the user can drag the correspondence points around to see the effect. Due to the global nature of the vector field color transformation, it is very easy to color balance an image to exactly match a photograph with similar content. Only a few references are needed to generate a close similarity between two images. Figure 10 shows the balancing of an underexposed image of the Hong Kong skyline using a photograph of the Golden Gate Bridge. The images in the figure show the step-by-step balancing, where the first two correspondences are placed on the sky (orange and green circles), and the second two on the water (yellow and red circles). Although only four points were used, the Hong Kong photo

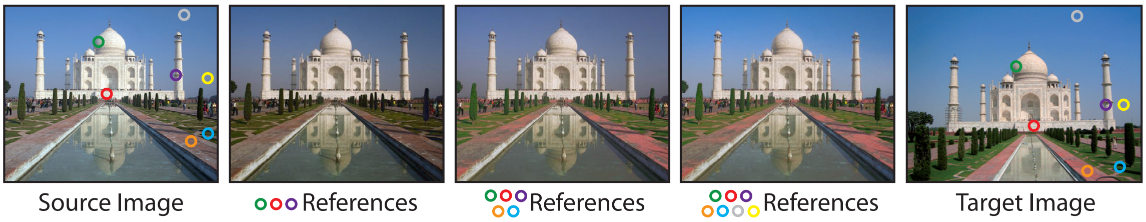


Figure 11: *An example of color balancing a photograph using another image from the same scene. The first three references match the colors of the facade. The next two color references are used to match the color of the faded red stones and the grass. Finally, the color of the sky is balanced using the last two references.*

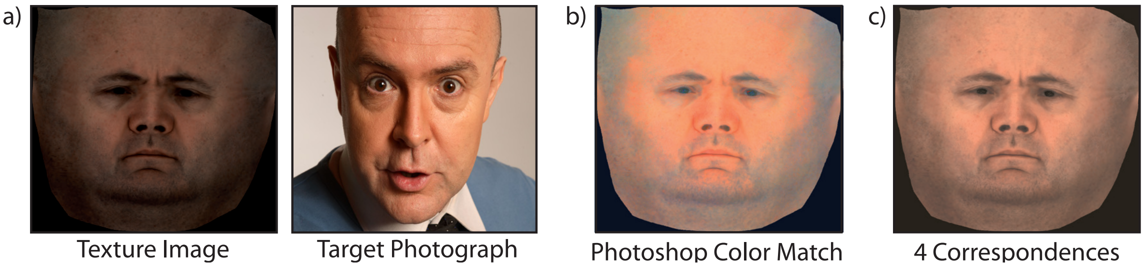


Figure 12: *This figure shows our color balancing method applied to texture correction. Often, in the process of face capturing, the colors of the reconstructed texture are not matching the original face. a) The texture from the face capture is shown together with a professional shot of the person that was scanned. b) Photoshop Color Match, that automatically corrects the colors in a source image given a target image, fails to correctly recreate the look of the photograph. Using our approach, the texture is balanced by using four color references (eye, lip, nose, forehead).*

quality improves significantly. Note, how both the water and the sky colors in the Hong Kong image adopt the exact colors of the Golden Gate photograph.

Due to the nature of our approach, it is very easy to match two images where multiple portions are supposed to have the exact same color. Only a few intuitive clicks are needed to generate close correspondences between the two images. An example is shown in Figure 11, where two different photographs of the Taj Mahal are balanced. The two images were taken with different cameras at different times. Note, that the source image (leftmost image in Figure 11) has different colors and the red stones bordering the water are faded. Using seven references, the source image is balanced to the target image. Even the faded stones look fresh again.

Another field of application of our color balancing approach is balancing texture colors from 3D scanners. Often, these multi-camera rigs are designed

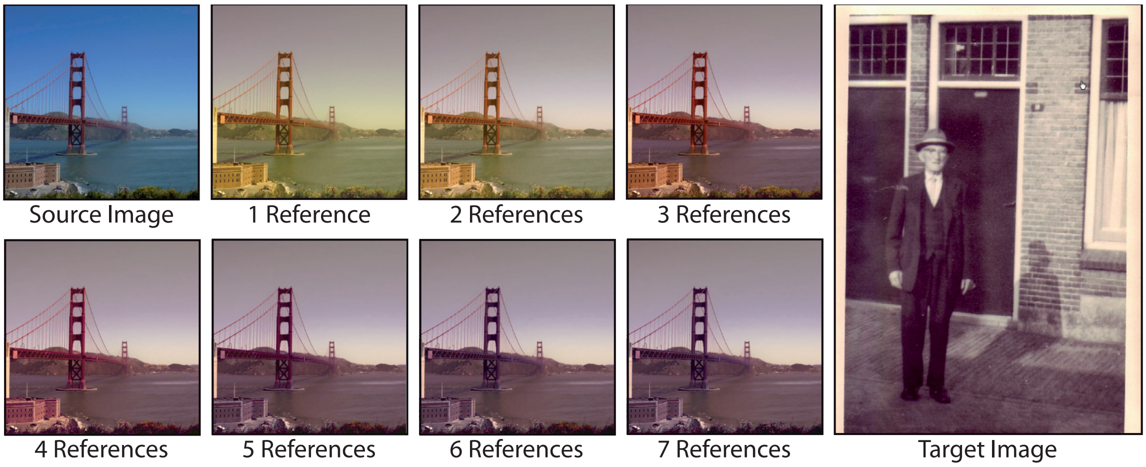


Figure 13: *An example of color-style transfer between two images with completely different content. The original color image of the Golden Gate Bridge on the top right is changed to match the old image of a gentleman on the right. With increasing number of correspondences, the image of the bridge changes its color style. With only 7 correspondences, the image matches the color palette of the target image.*

to best capture the geometry of the target object. It is usually a secondary task to create a texture for the geometry, and often the colors are incorrect. Figure 12 a) shows an example of a texture from a face scan compared to a professional photograph of the person. The colors of the texture have clearly undergone a change and it is desirable to adapt the colors in the texture image to match a professional photography. There are automatic methods that are able to match the colors of an image given a target. Figure 12 b) shows the result of the Photoshop Color Match function applied to our example. It is clearly visible that the colors of the jacket, for example, are also integrated into the texture (mostly re-coloring the beard). Also the specular highlights on the skin in the target image falsify the result. Our approach, however, is more suitable for such a case, allowing a simple selection of the desired colors in the target image to be matched to the source image. Figure 12 c) shows the result, where only four references are used, namely on the eye, lip, nose, and forehead.

Since our vector space color transfer is designed to handle non-linear color mappings, it is not only useful for balancing images, but, in fact, it is able to create extreme color changes as well. With only a few correspondences, color-styles of an image can be changed dramatically. Figure 13 shows an example where a color image is changed to the color style of an old photograph. The iterative nature of our application is beneficial in this case, where the style transfer mainly is user driven, as there is no single valid mapping.

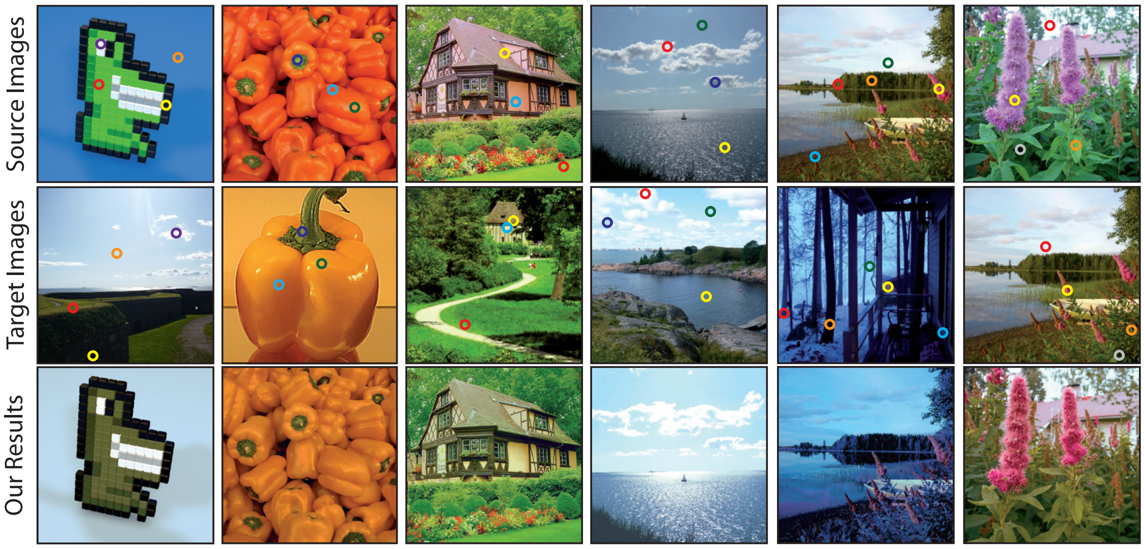


Figure 14: *Our sparse correspondence balancing allows a wide range of applicability. The top row shows the source images. The middle row contains the target images. In both images, colored circles denote color correspondences. The row on the bottom shows the balanced versions of the source images.*

The ability to completely change the color style of an image combined with the real-time implementation of our method allows to create a wide variety of example images without much effort. In Figure 14, a couple of examples are shown to demonstrate the wide range of applicability.

Figure 15 shows a series comparisons of our color balancing method with other color balancing approaches. Since our method is designed and optimized for global color balancing, it is able to transfer the colors from the target image to the source image with only a sparse set of references. User edits in Photoshop changing indirect parameters (hue, saturation, etc.) can be unintuitive and time consuming, and it is difficult to achieve exact results. Achieving the correct balance between the yellow and red colors in the bottom example are difficult with only indirect manipulation. Both the Photoshop Color Match function and the approach of Reinhard et al. [2001] perform very similarly. Both methods automatically achieve a global balancing of the image but cannot recreate the exact shade of the Taj Mahal. Using the sparse input used for our color balancing, the approach of Li et al [2010] struggles, as their method is specifically designed for local edits. In areas with strong gradients (e.g. the dome of the Taj Mahal) their approach produces artifacts. Linear regression is, due to its limited flexibility, not able to satisfy the constraints.

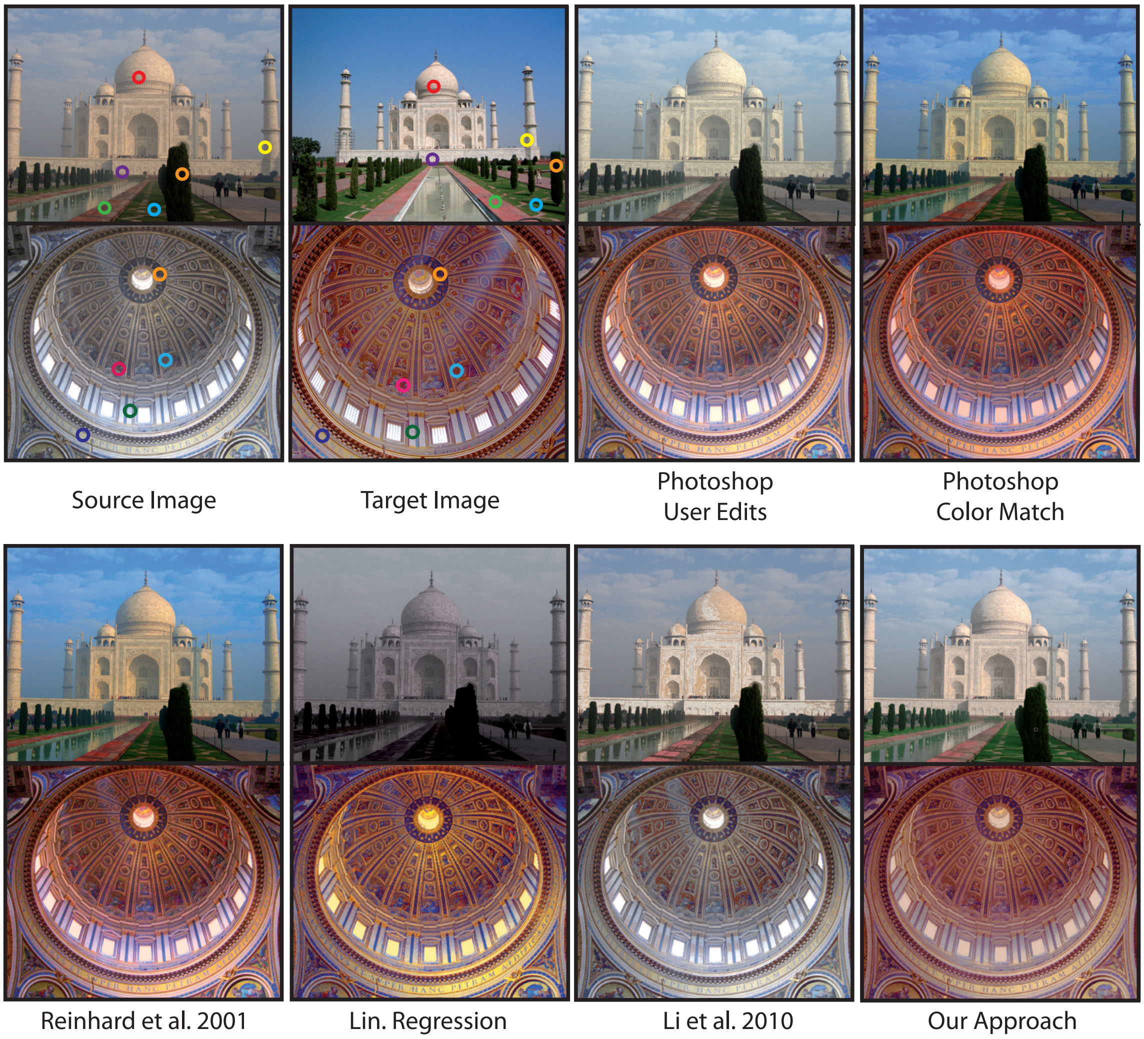
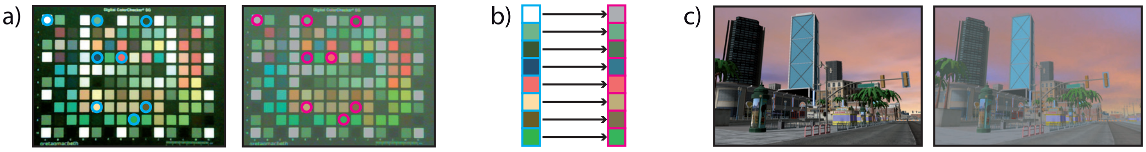


Figure 15: *Comparison of our approach with other methods for image balancing using sparse correspondences.*

4.2 Reproduction of Unknown Camera Color Transfer Functions

Our RBF color balancing framework allows to capture the behavior of a color change in an image by extracting color references. These color references are defined in the color space, and not the image space. Thus, they are independent of the image content. This property allows to apply the color transfer to different images without loosing the characteristic of the color change. Our color balancing thus allows the efficient and simple reproduction of camera color transfer functions only using two example images.

In order to achieve realistic color effects, a color checker pattern (in our case Digital ColorChecker SG) can be photographed using different settings of



Default Image Distorted Image Extracted Color References New Rendering Reproduced Color Transfer Function

Figure 16: *The pipeline for extracting an image capturing effect from example images and applying it to a new input image is shown. a) First, one neutrally recorded frame of a color checker and one frame with the desired color changes are selected. b) Then, colors from both image are extracted creating a list of color references. c). Finally, the color references can be used to balance a new input image in order to reproduce the capture effect.*

brightness, contrast, gain, etc. An example is shown in Figure 16. First, one frame is selected that shows the color checker under default parameter settings. Then, for each effect, a frame that shows the desired color changes is selected to create color references. In our experiments we have selected the colors indicated with the blue and pink circles in Figure 16 a). These references (shown in Figure 16 b) can now be used to balance a new input image to reproduce the color transfer function from the example images (Figure 16 c).

In addition to the simple but powerful color transfer function reproduction, our RBF color balancing approach has two distinctive advantages. First, a color transfer can be applied to new images very efficiently. Because the color references are only extracted once, and stay constant throughout the application, the most expensive part of the RBF computations can be omitted, namely the solving of the equation system to minimize overlapping support. The system only needs to be solved once during the extraction of the references. This allows the application of an image effect below one millisecond per frame. Second, our color balancing is designed as a vector field approach, effectively advecting the color gamut of an image. These vectors define the movement of each color in the color space. Therefore, a color transfer can actually be smoothly blended in by interpolating along the vector field without creating any overhead. Such a computationally cheap blending is not possible with classical approaches such as regression or histogram matching. Figure 17 shows some examples of effects applied with different amounts of blending to an input image in comparison with the ground truth recordings from the color checker. It can be seen that the blending is able to plausibly reproduce the color transfer function, even in between the examples. Note, that all the color checker images are photographs.

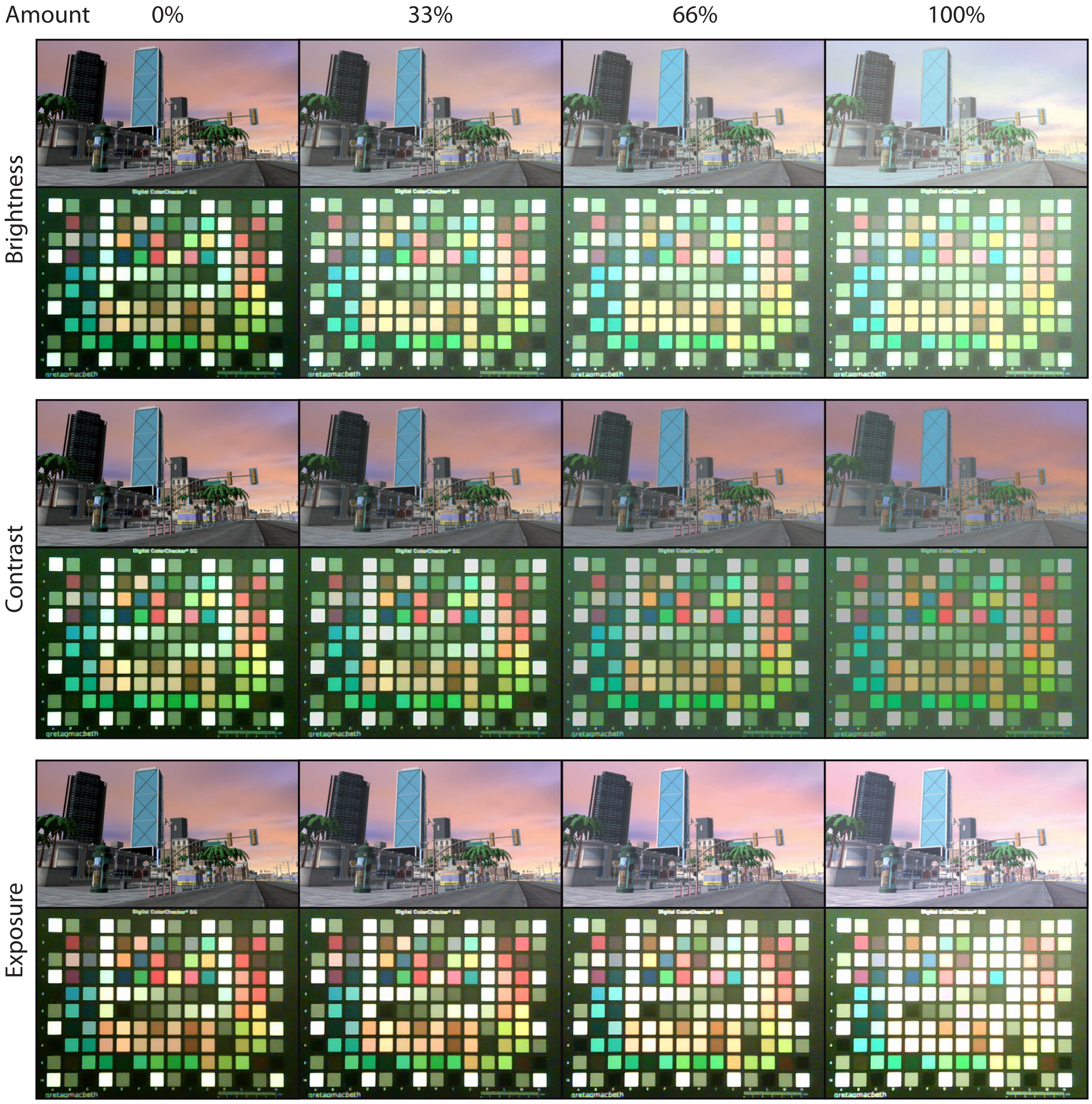
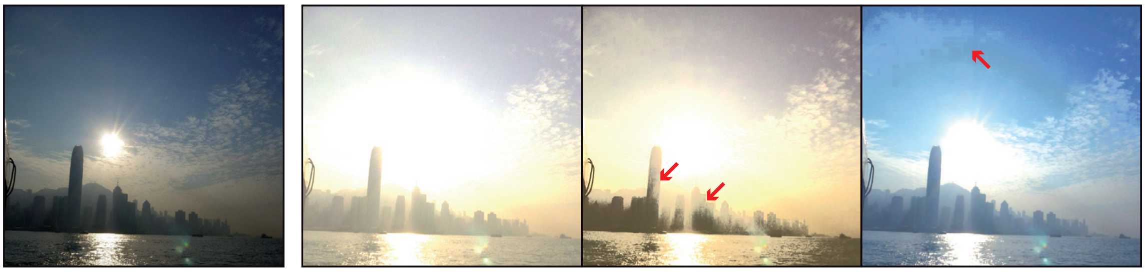


Figure 17: *A series of images showing the continuous reproduction of some camera color transfer functions. In each row, the image on the left is the original shot. Each image further to the right shows the successively increased application of a camera effect using our RBF color balancing, where the intermediate images are linear interpolations along the color transfer vector field. The color checker photographs are the ground truth images.*



Source Image

Overbrightening

Stepping Artifacts

Enhancing Compression Errors

Figure 18: *Several examples of errors caused by bad color references. Over-brightening is caused by constraining a source color with a bright target color. Stepping artifacts appear when over- or under-exposed areas are referenced with multiple different colors. In areas with a smooth color gradient, like skies, using several color references can cause image compression artifacts to appear more visibly.*

4.3 Color Balancing Limitations

In the previous sections we have proposed a fast and flexible method for color balancing based on vector field transformation. Using normalized RBF interpolation with optimized basis functions we are able to create color space transformations with only a few references. We have demonstrated the applicability of our method as an intuitive color balancing tool, for style transfer between images, and as an approach for reproducing color transfer functions of a digital camera.

Our approach, however, has a couple of limitations. While our color balancing is flexible and robust, bad correspondences can still cause artifacts. Figure 18 shows some of the problems. Over-brightening is caused by referencing a source color with a bright target color. In this case our algorithm transforms the image gamut partly outside the displayable color range. These colors are then clamped to the display gamut, causing over-brightening. The same effect appears with dark colors as well. Furthermore, our color correction is not able to reconstruct missing information. Over- or underexposed images can flatten color gradients to the point where only a few colors remain. Our color transfer method does not smooth these gradients spatially, and stepping artifacts appear. Similarly, when referencing multiple colors in areas of gradients, compression artifacts may become visible. The former two problems could be attacked by adding a spatial component to the RBF interpolation that takes the image location of the pixel into account.

To reproduce color transfer functions using example photographs from a color checker pattern, we currently select the set of color references by hand. Using base colors such as red, green, blue, etc., we are able to create convincing results. However, it would be desirable to select a suitable set of color references automatically. To achieve this, the images used as examples could be analyzed and compared to the reproduced results in order to find the references with the highest impact.

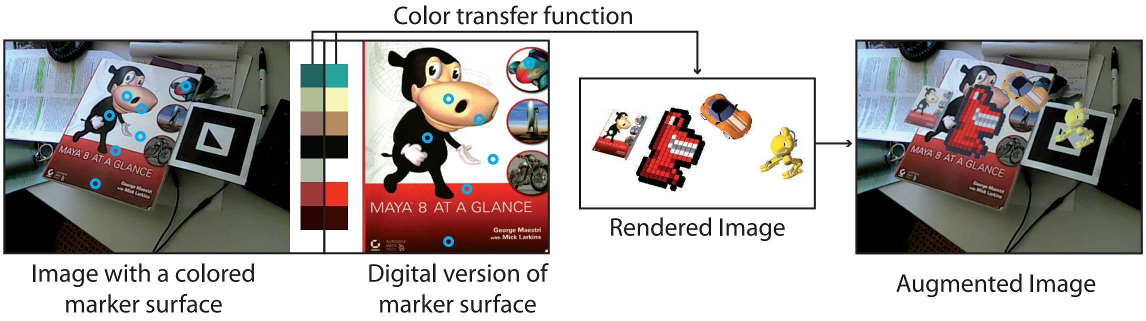


Figure 19: *The general idea of our approach to automated color correction in augmented reality applications. A known surface (like the cover of a book) is filmed, and the colors of positions on that surface are tracked and compared with the colors of a digital representation. These color references, dynamically acquired each frame, allow the balancing of rendered images to the video stream using our RBF color transformation method. This creates a balanced and adaptive composition of virtual and real images.*

5 Extension to Augmented Reality

A typical scenario where the permanent update of the color balance improves the visual quality is augmented reality, where a synthetic rendering should be embedded in a seamless manner into a video stream. In order to tackle this problem, we propose to use known color patches in the scene (see Figure 19). If these colors can be robustly tracked over time, they can be used as color references for their supposed values. These references then automatically balance rendered images to the video footage, adapting the renderings as the color of the video frames change due to camera adjustments.

There is previous work that looks at different aspects of improving augmented images, like noise emulation or lens- and motion blur estimation [Park et al., 2009; Klein and Murray, 2010]. In contrast, we are focussing on the problem of estimating the camera color transfer function at run-time. In order to capture the color distortions caused by changing camera settings such as exposure or gain, we need to establish color correspondences between video frame and rendered image. These correspondences can then be used to transform the entire color-space of the rendered footage such that it appears to be filmed by the camera. Note that we are not trying to estimate global lighting, as this is a different research topic all together. Moreover, we propose to add another layer of realism on top of properly rendered lighting.

In order to robustly acquire color references on a per-frame basis it is crucial to remove corrupt colors caused by occlusions or other external effects such as specular reflections. The difficulty lies in the fact that color changes may be also desired, as in the case of the camera automatically adjusting its internal parameters. In this section we will discuss an approach to separate external and internal color changes, and take further measures to obtain a stable set of color references over time before we show some results.

5.1 Separation of Internal and External Color Changes

The first task is to track a set of known colors within the video stream. Tracking the surface position itself, and thus individual color sampling positions on the calibration image, can be performed out of the box using existing tools. There is a large variety of methods, some of which use markers [Liere and Mulder, 2003; Lee et al., 2010], some use geometry [Drummond and Cipolla, 2002; Lepetit and Fua, 2006; Klein and Murray, 2006], and some use surface or textures features [Lowe, 2004; Benhimane and Malis, 2006; Klein and Murray, 2008]. We therefore assume the position tracking to be a black-box process that is provided to us. In our implementation we use ARToolKit marker tracking [ARToolKit, 2012].

We can assume that we know the locations of the pixels in each frame that correspond to positions on our digital marker version (see Figure 19). We assume no knowledge on the internal camera parameter settings (like exposure or gain). The goal is to find out which positions in the video stream contain desired colors (i.e. only affected by internal color changes), and which positions contain corrupted colors (i.e. affected by external color changes). The following points have to be considered:

Changing Camera Parameter: Depending on the scene, the camera usually adjusts its internal parameter (exposure, gain, contrast) over time. In general, manufacturers of these cameras do not specify the algorithms used to change these parameters. Even worse, the parameters themselves are usually not available, making it more difficult to reproduce the color transfer. Previous approaches have generally disabled these parameter [Klein and Murray, 2010]. However, we want to provide a solution usable in uncontrolled settings, where parts of the camera cannot be disabled.

Occlusions: Tracked colors can be occluded by an object in the scene (like waving ones hand in front of the camera). This needs to be detected and the color removed from the list of correspondences. Otherwise, the transfer function becomes erroneous and can produce false results.

Specular Reflections: Specular highlights caused by light sources in the scene can occur on the calibration surface. They are usually smooth and change the color of the tracked position gradually over time to a bright white. Similar to occlusions, such changes need to be detected in order avoid artifacts.

Inaccurate Positional Tracking: Because position tracking is treated as a black box, there is no knowledge on the accuracy of the method. The tracking may be noisy or can even fail entirely at times. Our method needs to be robust against this noise, and must be able to cope with temporal failures of the tracking. Moreover, these failures need to be accounted for by providing transition correspondences in order to prevent the color transfer from starving on too view correspondences.

Per-frame processing: In order to be usable in augmented reality applications, no assumptions can be made about the future behavior of the camera. In a real-time application, a method must function on-line with as minimal of temporal delay as possible. Therefore, the approach has to be limited to a per-frame solution, that at most uses information from the past to solve problems in the present.

Observations and Transformation Model

Having specified the requirements, we can now look at ways to implement such a color tracking. Only the video stream and the position of a known marker in the scene are available. Using only this information we want to provide a series of color correspondences that captures the color transfer function of the camera. The goal is to provide these pairs of colors for each frame in order to instantly react to changes of the camera internal parameters (e.g. exposure, gain). A solution to the problem is to fit an appropriate model to the tracked color values that predicts what colors have undergone an undesired change versus a desired change. Let us consider the following observations to find such a model.

Let us first look at an example of how the La^*b^* color space profile changes when the internal camera parameters are changed. Figure 20 shows a series of stills from this experiment. Even though the exact functions are not known, it can be observed that the color space profile seems to undergo a global transformation.

Another scenario is the color change of points when they are occluded. Figure 21 shows the color profile for several degrees of occlusions of the tracked surface. While colors that are not occluded (or not affected by shadow changes) naturally remain constant, colors of occluded points group together in the area of the occluder's color. In Figure 21, this is indicated by the orange mask. These changes in the color profile are non-linear and local. This is in contrast

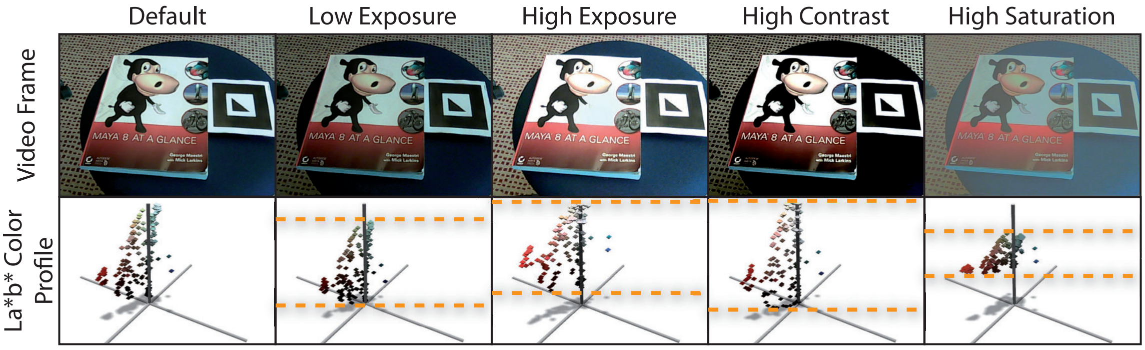


Figure 20: *Changes in the La**b* *color space profile of a tracked surface (book cover) when internal camera parameters are adjusted. The top row shows frames from a static scene that is filmed under different camera parameter settings. The bottom row shows the La**b* *color space profile of 196 uniformly distributed points on the surface. Even though the exact transfer function is not known, it can be observed that the color space profile changes globally.*

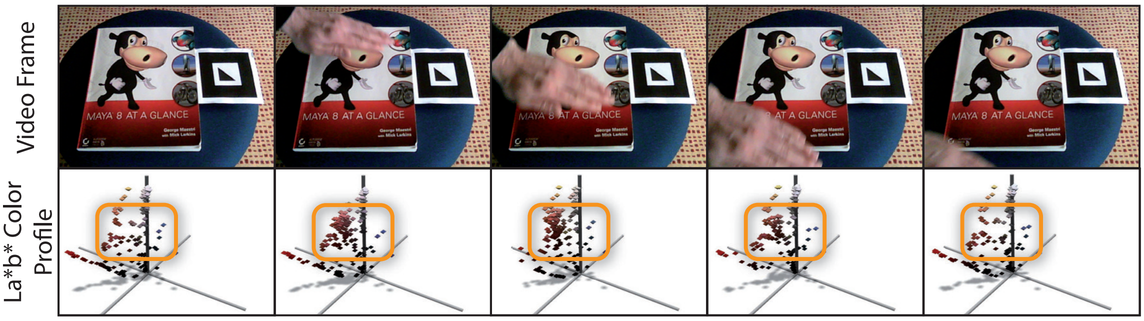


Figure 21: *Changes in the La**b* *color space profile when parts of the surface are occluded. It can be seen that occluded points cause a higher density in the La**b**-profile in the masked region. This is a local non-linear change of the color profile.*

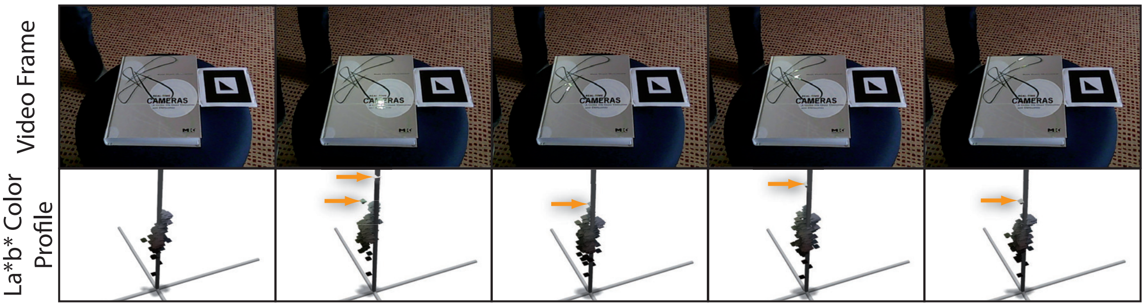


Figure 22: *Changes in the La**b* *color space profile with specular highlights on the surface. The thin specular highlights cause the tracked color points to become significantly brighter. The orange arrows show instances of such colors. In this example only individual colors are changing as the specular highlight is very small compared to the surface.*

to the camera parameter changes, that caused a global transformation of the entire color profile.

Similar to occlusions, specular reflections may occur and completely cover a portion of the tracked surface. These need to be detected as well. Figure 22 shows an example of small specular highlights on a partly reflective surface. The specular reflection causes a tracked color point to become gradually brighter. The Figure shows that these changes affect the color of individual points nonlinearly. This is similar to the observation of the color changes caused by an occlusion.

The experiments shown in Figures 20, 21, and 22 indicate, that undesired changes may differ, but each show a significantly different behavior of the color profile than the desired changes. Similarly, tracking noise that lets a point travel across a color boundary shows a similar non-linear and local behavior.

Therefore, in order to separate internal and external color changes, we can fit a global model that ties the general behavior together. Desirable in our case is a model that (i) is very fast to compute, as it might need to be re-fitted multiple times, and (ii) is not over-fitting the data. It is, in our case, more desirable to reject some good colors than to accept corrupt ones. Rejection of good color points can be countered by adding redundancy.

A good model choice to estimate the global change of a set of colors from one frame to the next is the affine transformation [Adams et al.,]. It is flexible in a global sense but is robust enough to prevent over fitting. Additionally, global affine transformation can be computed analytically, and is therefore very cheap in terms of computational overhead. Given a list of color pairs (c_i, d_i), the affine transformation matrix A and translation t that transform the c_i as close as possible in the least squares sense onto the d_i can be computed as

$$A = \sum_i \tilde{d}_i \tilde{c}_i^T \left(\sum_i \tilde{c}_i \tilde{c}_i^T \right)^{-1} \tag{12}$$

$$t = \hat{d} - A\,\hat{c}, \tag{13}$$

where $\hat{c}$ is the average of all c_i, and $\tilde{c}_i = c_i - \hat{c}$. $\hat{d}$ and $\tilde{d}$ are computed analogously. A detailed derivation of the affine transformation model can be found in Appendix A.

We have performed several tests on the performance of the affine transformation model on a color checker pattern. Figures 23 and 24 show how the

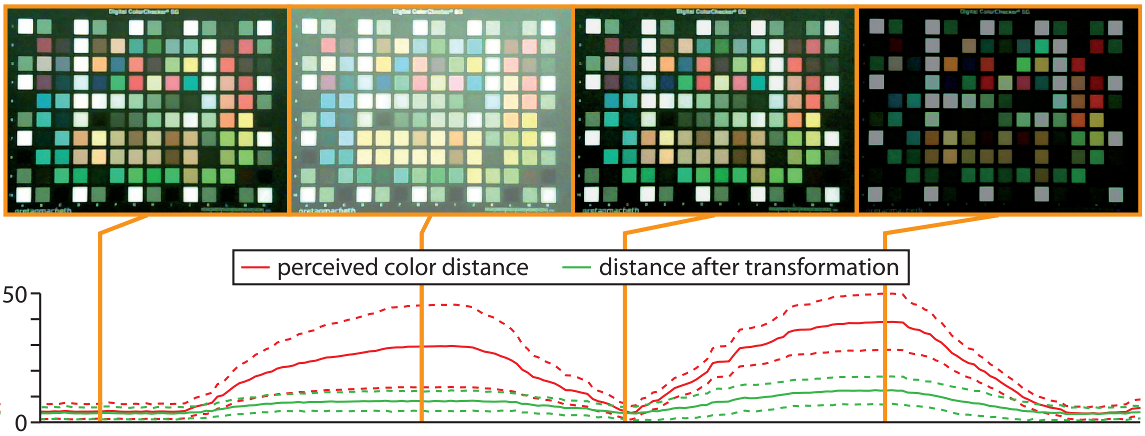


Figure 23: *Color distances under changing brightness setting of the camera. The red graph shows the average distance in La*b* space between the start image and the image recorded under a different brightness setting. The green curve shows the distance after the optimal affine transformation has been fitted. The dashed curves show the standard deviation. It can be observed that fitting an affine model reduces the amount of measured color distances considerably, reducing the color change between consecutive frames.*

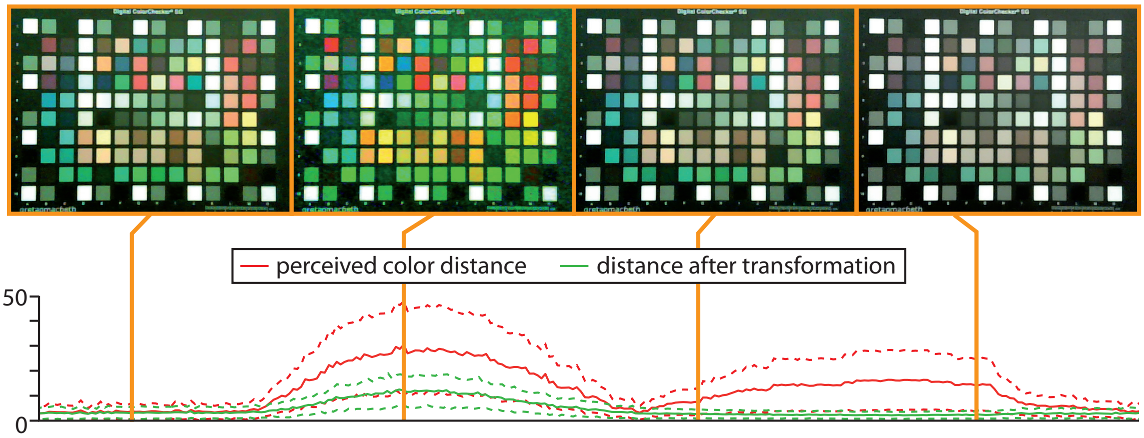


Figure 24: *Color distances under changing the image intensity setting of the camera.*

perceived distances are reduced after fitting and subtracting the affine model to images with changing brightness and intensity. Other image parameters such as gain and exposure showed similar results. The tests confirm that the affine model is able to reduce the global color change measured from one frame to the next. However, the results also show, that the model is not able to reconstruct the colors exactly. Nonetheless, the affine model is able to reduce most distance below 10 units in the La^*b^*-space. This value gives an indication on the threshold value that should be used to detect local outliers when fitting an affine model.

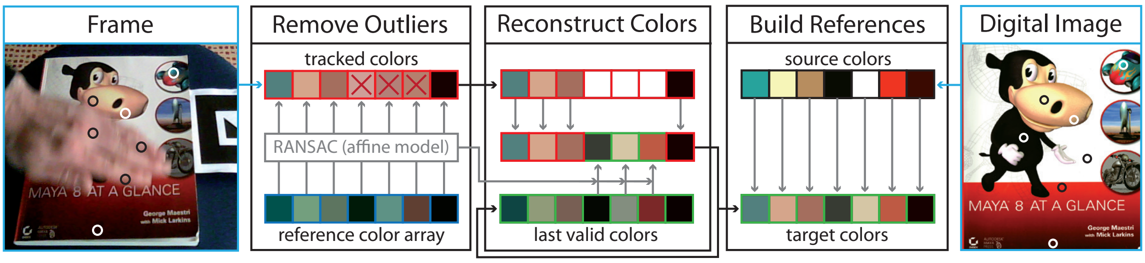


Figure 25: *Overview of our approach to color tracking. From left to right: First, the colors from the tracked positions are extracted and compared to reference colors. Using the RANSAC algorithm an affine transformation model is fitted to remove outliers. Then, using the inliers and last valid colors, missing colors are reconstructed.*

5.2 Robust and Stable Color Tracking

In the following, we explain our approach that automatically separates undesired occlusions, pattern misalignments, and specular highlights on the pattern from desired color changes caused by camera parameter adjustments. First, an initialization needs to be performed. Then, the run-time algorithm is applied each frame in order to remove outliers and reconstruct a stable set of colors. An overview is given in Figure 25. For each frame, the following steps are performed:

1 Extract Colors. The first step is to extract the colors d_i from the video frame. The position tracking provides the positions in the image where the target colors are found. To reduce per-pixel noise we perform a neighborhood averaging over a small window (usually 7 by 7).

2 Detect Outlier. To detect corrupted colors we can fit a global model between the extracted d_i and a set of reference colors. These reference colors r_i can be taken from the digital image of the marker or extracted from the first frames of the video in a pre-processing step. In order to keep the number of false positives low a conservative model should be fitted between the c_i and r_i. We chose the affine transformation model $c_i = \mathbf{A} r_i + \mathbf{t}$ [Adams et al.,], which can be computed analytically and avoids over fitting to corrupted colors.

Outlier in the scene colors d_i can now be detected by fitting the affine model using the random sample consensus algorithm (RANSAC) [Fischler and Bolles, 1981]. RANSAC functions by randomly selecting the minimum amount of points needed to fit a given model and then measure how close all other points are approximated by the fitted model. The points that fall outside a certain treshold are considered outliers, and the model is accepted if the number of outliers is smaller than for every other model fit before. This process is repeated enough times to statistically provide a 99% chance that no outliers are in the group of points initially selected.

In our implementation we are using RANSAC by picking randomly four pairs from the set of colors (c_i^R, c_i). We then compute the matrix A according to Equation 12 and the translation vector t according to Equation 13. After fitting the affine transformation model to the subset of color pairs all remaining reference colors c_i^R are transformed by A and translated by t. Now, the distance r_i between each transformed reference color c^R and the corresponding color c from the video frame is computed the distance r_i is computed.

This first step allows removing most of the corrupted colors. However, false positives may slip though this outlier detection. In order to further increase the robustness of the tracking, we separate the remaining inliers into two groups. Trusted inliers are colors that have not been detected as outliers for more than κ_t frames. These colors are used as valid target colors. Inlier colors that have been detected as outlier in one of the past κ_t frames may be false positives. These colors are also treated as outlier. In our experiments we found that false positives rarely appear for more than two frames. Therefore, in our experiments, we mostly used $\kappa_t = 3$.

3 Reconstruct Colors. At this point, corrupted colors have been removed from the list of d_i. Additionally, the risk of false positives has been decreased by taking temporal consistency into account. Now, in order to maximize the amount of colors available for color balancing, we can use the last valid colors of each tracked point from the previous frames. These colors are stored when a tracked value is regarded as trusted inlier (being not an outlier for more than κ_t frames). Colors values, that are suddenly corrupt due to occlusions or specular reflections can then be replaced by an updated version of their last valid value. The update of these last valid colors is performed by applying the global affine transformation performed by all trusted inlier colors since the last frame. In our implementation, we perform this color reconstruction only for outlier that have been inliers for more than κ_c frames. This removes colors that were only detected as inliers for a short period and increases the robustness. We call these colors comeback points. In our implementation we have set $\kappa_c = 10$.

4 Create References. The final color references (c_i, d_i) for the balancing of rendered objects can now be created. For all d_i, that are either trusted inlier or a comeback points, the corresponding color c_i can be extracted from the digital image of the marker. The constraints (c_i, d_i) now describe the color transfer function from rendered footage to the current video frame color space.

With our fast global color balancing for sparse color correspondences we are able to adapt the colors to fit to a target image. With our robust color tracking, we can extend this into the temporal domain. Using a known marker in the scene (e.g. the cover of a book) we can balance newly rendered footage at to augment the video and increase the realism of augmented reality applications.

5 Update Reference Colors. The outlier detection in the first step relies on a set of reference colors to which we compare the extracted colors in the current frame. A good reference is, for example, a clean photograph of the scene. Also a digital version of the colors (i.e., a scan of the image) that should be observed can be used for extracting reference colors.

In each frame, we use trusted inliers as new reference colors in the next frame, effectively replacing the old ones. The rest of the reference colors are transformed according to the best fitting affine model (A, t) found during outlier detection.

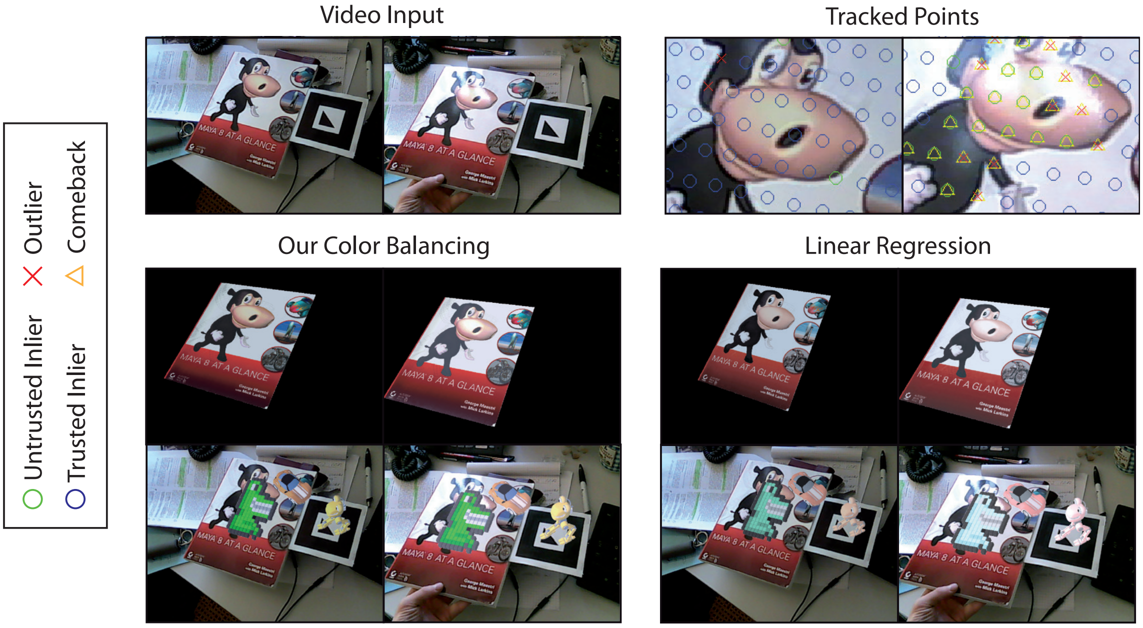


Figure 26: *An example of our color balancing compared to linear regression balancing in augmented reality applications. Outliers (red x) arising due to specular reflection are detected and supported with comeback colors (yellow triangle). Our approach is both more accurate in recreating the exact image colors as well as produce more plausible color balancing for unknown colors (i.e. green frog) than linear regression.*

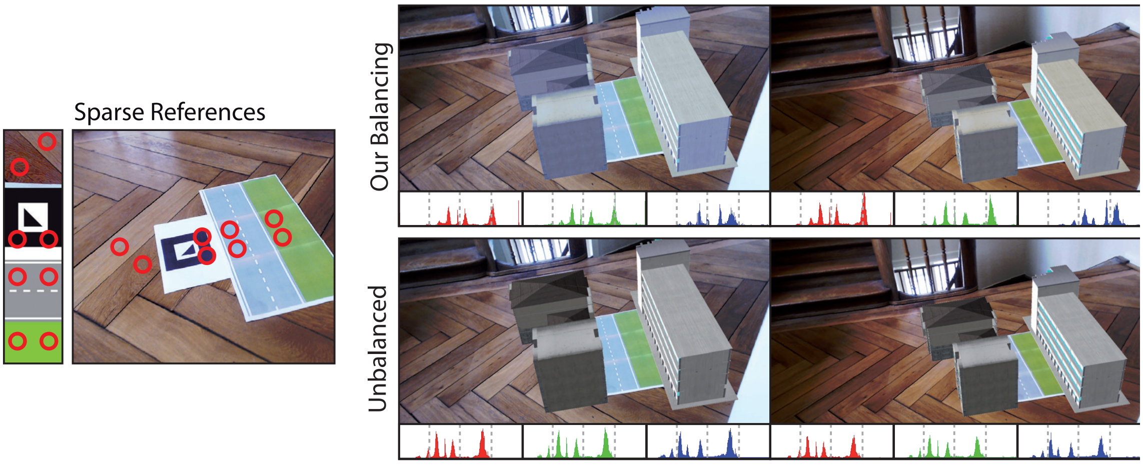


Figure 27: *A comparison where only sparse color samples are tracked is shown. Our color balancing clearly adjusts the colors to the fames. This can also be verified by comparing the change of the color histograms between the frames.*

5.3 Results

While our color transfer approach only utilizes the flexible vector space color transfer, applications in augmented reality become possible when combining it with our robust color tracking. A known marker in the scene, like the cover of a book, can be tracked and the colors referenced with a digital copy of the image.

Figure 26 shows a comparison between our color correction and linear regression. It can be observed that our approach is more accurate for colors that are available in the video stream and more robust for colors that are not tracked.

The example shown in Figure 27 depicts frames in comparison where only sparse color samples are tracked. Even with only 5 color references available, our approach is able to emulate color changes due to camera parameter adjustments for rendered objects. By comparing the change of the color histograms (they only count colors from the rendered portions of the image) between the two frames it is apparent that our example reacts to the video stream, whereas the histogram peaks from the unbalanced rendering stay at the same positions.

The example in Figure 28 shows an application of our method where an animated image is placed in a book. Due to the color tracking of the real image on the same page we are able to embed the animated figure into the video more realistically. Note that the adjustment of the colors in the flat shaded examples by hand would only work for one frame as the white balance and global light change over time.

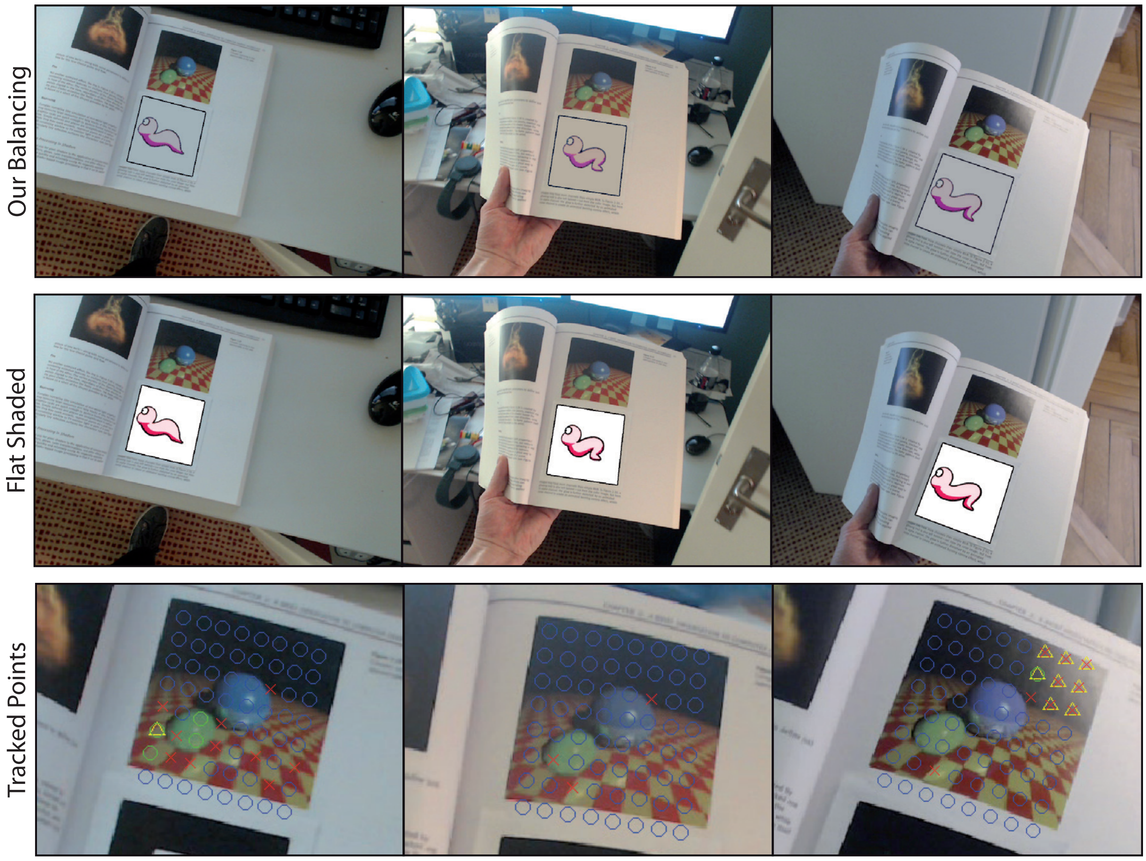


Figure 28: *An example application of our method is the augmentation of an animated image in a book. Our method is able to increase the realism by properly balancing the images to the changing colors of the video stream.*

5.4 Limitations

The proposed tracking and correction algorithms have some limitations that direct us to areas of future work. The color tracking algorithm relies on flat color areas in the image to work well. This requirement is orthogonal to many tracking algorithms that use features like corners or edges. However, our algorithm will reject such colors instead of producing false results. In our current implementation we employ equidistant sampling of points on the marker surface where the colors near features get removed by our tracking. Improving this sampling using information such as edges or gradients in the marker image would be an interesting topic to explore.

The current implementation only provides a tool for tracking and balancing of colors. The choice of the marker surface, however, also influences the quality. Not any surface is suitable for color tracking due to high-frequency content or lack of variety in color. A particularly exciting future venue would be to investigate the creation of good marker by changing existing surfaces (e.g. the cover of a book) slightly for better performance. Also the seamless integration of such marker into the environment would be an interesting topic.

To this end, the extension to augmented reality is an interesting application of our fast color balancing approach. However, the current implementation has only been tested in a controlled lab environment. Although the examples seem to work well, one venue for future would be to perform a proper analysis and collect statistics of our tracking and balancing in realistic settings. Insights from such tests could improve the understanding of the robustness and flexibility of our tracking could ultimately lead to the ability to apply or adapt our method to commercial settings.

CHAPTER 6

Conclusion

We conclude this dissertation with a summary of our contributions and a discussion on future research.

Contributions

In this thesis, we visited three important areas of virtual camera control. We have discussed problems of three different levels, namely motion and view planning for large scale camera control, automatically optimized stereoscopy, and the simulation of internal camera behavior through vector space color balancing using example images. Throughout the works presented in this dissertation we have specifically targeted fast and efficient solutions for high-performance interactive applications such as computer games.

In Chapter 3, we presented an algorithm for visibility transition planning that can compute large, collision-free camera transitions in real-time in dynamic environments. This achievement rests on several key insights. We develop a visibility-aware roadmap data structure that allows the pre-computation of a coarse representation of all collision-free paths through an environment, together with an estimate of the pair-wise visibility between all portions of the environment. Once the start, end, and focus point have been specified, our runtime system executes a path planning algorithm using the precomputed roadmap values to find a coarse path that is optimal in terms of visibility up to the resolution of the roadmap. Next, the path is refined by computing a sequence of GPU-assisted occlusion maps along the coarse path. The same path-planning code is executed with these occlusion maps to enhance visibility on a fine scale. An iterative smoothing algorithm together with a physically-based camera model ensure that the path followed by the camera is smooth in both space and time. All run-time computation is output sensitive, so that the required time depends on the final path length. The visibility-aware roadmap data structure adapts dynamically to occluders that move in an environment, supporting opening and closing doors, falling boulders, and other occlusion situations in real-time.

In Chapter 4, we described an effective and efficient solution for optimizing stereoscopic camera parameters in interactive, dynamic 3D environments. On the basis of a viewer-centric and a scene-centric model, we have defined the mapping between the scene depth and perceived depth as an optimization problem. We have derived constraints for a stereoscopic camera controller that is capable of rendering any visible scene content optimally into any target depth range for arbitrary devices and viewing configurations. Moreover, we have addressed the problem of blending stereoscopic parameters and the resulting nonlinear distortions in perceived depth. Our method allows for a linearization of such effects, but also for more complex temporal transformations to render desired depth effects in the target space. With running times less than 0.2 ms per frame at full HD resolution, our controller is fast enough even for demanding real-time applications. Our experimental evaluation showed that our controller is preferred over naive stereoscopic rendering.

In Chapter 5, we have presented an approach for interactive image-based color balancing using only a sparse set of correspondences, as well as an extension for temporally consistent color correction for augmented reality applications. Through our proposed global optimization of basis functions, we provide a tool to optimize function parameters to mimic color changes from example images and optimally adjust colors accordingly. We have shown, that using this approach, camera color transfer effects such as adjusting gain, contrast, exposure, etc., can be easily reproduced with very little overhead to the rendering.

Future Research

Some limitations of our interactive camera control algorithms direct us to exciting future work.

Dynamic Visibility Roadmap. The current form of our visibility-aware roadmap is created for mostly static environments. With the extension to handle dynamic occluders we add the ability to handle the most common virtual environments, where most of the geometry is static. However, with the development of new interactive environments that allow for increasing degrees of destruction and geo-morphing, our data structure will become inefficient. An interesting future venue for our framework would be to investigate possibilities of dynamically updating the sphere-based roadmap to changing environments. While the update of the spheres regarding ambient space might be possible to achieve in interactive frame rates through iterative updates, updating the visibility information might be a greater challenge. A possible solution could be to generalize the occlusion map rendering, updating sphere visibilities by approximating the surrounding visibility through occlusion cube maps.

Combined Motion and Stereoscopy planning. Our optimized stereoscopic camera control for interactive 3D allows the ad-hoc optimization of the two basic stereoscopic parameters, the camera separation and convergence, in order to achieve an exact constraining of the perceived depth. We have shown that this control is able to lessen problems of exceeding disparities in environments where the camera cannot be controlled. The combination of this framework with motion control provided by our camera planning framework could open up interesting future work. Currently, camera motion control and stereoscopic parametrization are completely separated tasks. However, the positioning of the camera has also a significant impact on the perceived scene depth. Using this together with the optimization of stereoscopic parameters could lead to interesting new camera work enabling more cinematography oriented control.

Content- and Context-aware Image Balancing. Using our vector space color balancing, we are able to reproduce color changing behavior of real cameras that occur when the scene content changes. While our method is able to learn the shape of basis functions to optimally reproduce the non-linear nature of color transfer functions with only few references, it does not utilize image content nor the context in which the image was produced. This missing link, although allowing fast processing, limits the approach to color space transformations. In future research, a possible link to the content of the image and the circumstances of the image rendering could be made, directly linking the color balancing to the camera motion and view control. By incorporating environmental factors, camera color transfer effects could be simulated rather than reproduced. Furthermore, a back coupling of the image balancing to both the intrinsic and extrinsic parameter control could induce interesting new behavior, and converge the pipeline more towards a unified camera framework.

APPENDIX A

Derivations

Affine Color Transformation Given a list of color pairs (c_i, d_i) in a three dimensional color space, an affine transformation matrix A and translation vector t can be found such that the source colors c_i are transformed as close as possible in the least squares sense to the target colors d, minimizing the energy

$$E = \frac{1}{2} \sum_{i=1}^{n} ||A c_i + t - d_i||^2. \tag{1}$$

The affine transformation minimizing E can be computed analytically. In the following we derive the closed form for A and t.

First, let us define the per-color error as

$$e_i = A c_i + t - d_i. \tag{2}$$

Now, the derivative of Equation 1 with respect to t can be written as

$$\frac{\delta E}{\delta t} = \frac{\delta}{\delta t} \frac{1}{2} \sum_{i=1}^{n} e_i^T I e_i. \tag{3}$$

Equation 3 can be reformulated [Petersen and Pedersen, 2008], and set equal to zero.

$$\begin{aligned}\frac{\delta}{\delta t}\frac{1}{2}\sum_{i=1}^{n}e_i^T I e_i &= \frac{\delta}{\delta t}\frac{1}{2}\sum_{i=1}^{n}(I+I^T)a_i \\ &= \sum_{i=1}^{n}(A c_i + t - d_i) \\ &= 0\end{aligned} \tag{4}$$

Equation 4 can now be rearranged to

$$\begin{aligned}0 &= A\sum_{i=1}^{n}c_i + \sum_{i=1}^{n}t - \sum_{i}d_i \\ &= A\sum_{i=1}^{n}c_i + n t - \sum_{i=1}^{n}d_i.\end{aligned} \tag{5}$$

Finally, we can solve for t

$$t = \frac{1}{n}\sum_{i=1}^{n}d_i - A\frac{1}{n}\sum_{i=1}^{n}c_i. \tag{6}$$

Using the mean values of the source and target colors

$$\begin{aligned}\hat{c} &= \frac{1}{n}\sum_{i=1}^{n}c_i \\ \hat{d} &= \frac{1}{n}\sum_{i=1}^{n}d_i,\end{aligned}$$

we can rewrite Equation 6 to

$$t = \hat{d} - A\hat{c}. \tag{7}$$

Equation 7 is the closed form to compute the translation between the two sets of colors. Little surprising, it is equal to the difference between the transformed centroid of the source colors c_i and the centroid of the target colors d_i. However, Equation 7 still depends on A, which is currently unknown.

Now, let us go back to the energy function in Equation 1 and compute the derivative with respect to the affine transformation matrix A.

$$\frac{\delta E}{\delta A} = \frac{\delta}{\delta A}\frac{1}{2}\sum_{i=1}^{n}||e_i||^2 \tag{8}$$

In order to find the closed solution to Equation 8, we can extend it to

$$\frac{\delta E}{\delta A} = \frac{\delta}{\delta A}\sum_i a_i^T a_i, \tag{9}$$

and compute the derivative [Petersen and Pedersen, 2008] and set the term equal to zero.

$$\begin{aligned}\frac{\delta E}{\delta A} &= \sum_{i=1}^{n} e_i c_i^T \\ &= \sum_{i=1}^{n} (A c_i + t - \hat{c}_i) c_i^T \\ &= 0\end{aligned} \tag{10}$$

Equation 10 can now be rearranged to the following form

$$A\sum_{i=1}^{n} c_i c_i^T + t\sum_{i=1}^{n} c_i^T - \sum_{i=1}^{n} d_i c_i^T = 0. \tag{11}$$

In order to find the closed form for A that is independent of the translational part, we can employ a little trick to remove t from Equation 11. In Equation 7, t only depends on the mean color values $\hat{c}$ and $\hat{d}$. So if we move all color values in such a way that these mean values become zero, we can eliminate t.

$$\tilde{c}_i = c_i - \hat{c}_i \tag{12}$$

$$\tilde{d}_i = d_i - \hat{d}_i \tag{13}$$

The replacing the c_i with $\tilde{c}_i$ and the d_i with $\tilde{d}_i$ causes the translation t to become zero, and Equation 11 is reduced to

$$A\sum_i \tilde{c}_i \tilde{c}_i^T - \sum_i \tilde{d}_i \tilde{c}_i^T = 0. \tag{14}$$

Now, we can solve for the closed form of the affine transformation matrix

$$A = \sum_i \tilde{d}_i \tilde{c}_i^T \left(\sum_i \tilde{c}_i \tilde{c}_i^T \right)^{-1}. \tag{15}$$

Equations 7 and 15 enable to analytically compute an affine transformation matrix A and translation vector t that transforms a set of colors c_i as close as possible in the least squares sense to a second set of corresponding colors d_i.

Bibliography

[Abadpour and Kasaei, 2004] A. Abadpour and S. Kasaei. A Fast and Efficient Fuzzy Color Transfer Method. In *Signal Processing and Information Technology*, pages 491 – 494, 2004.

[Adams et al.,] Jim Adams, Ken Parulski, and Kevin Spaulding. Color Processing in Digital Cameras. *IEEE Micro*, 18(6).

[Agarwal et al., 2006] Vivek Agarwal, Besma R. Abidi, Andreas Koschan, and Mongi A. Abidi. An Overview of Color Constancy Algorithms. *Journal of Pattern Recognition Research*, pages 42–54, 2006.

[An and Pellacini, 2008] Xiaobo An and Fabio Pellacini. AppProp: All-Pairs Appearance-Space Edit Propagation. *ACM Transactions on Graphics*, 27(3), 2008.

[An and Pellacini, 2010] Xiaobo An and Fabio Pellacini. User-Controllable Color Transfer. *Computer Graphics Forum*, 29(2):263–271, 2010.

[ARToolKit, 2012] ARToolKit. Software Library for Building Augmented Reality Applications., 2012. [Online; accessed 29-June-2012].

[Backus et al., 1999] B. Backus, M. S. Banks, R. van Ee, and J. A. Crowell. Horizontal and Vertical Disparity, Eye Position, and Stereoscopic Slant Perception. *Vision Research*, 39(6):1143–1170, 1999.

[Bandyopadhyay et al.,] Tirthankar Bandyopadhyay, Yuanping Li, Marcelo H. Ang Jr., and David Hsu. Stealth Tracking of an Unpredictable Target Among Obstacles. In *Algorithmic Foundations of Robotics VI*, pages 43–58.

[Bandyopadhyay et al., 2006] Tirthankar Bandyopadhyay, Yuanping Li, Marcelo H. Ang Jr., and David Hsu. A Greedy Strategy for Tracking a Locally Predictable Target among Obstacles. In *ICRA*, pages 2342–2347, 2006.

[Bandyopadhyay et al., 2007] Tirthankar Bandyopadhyay, Marcelo H. Ang, and David Hsu. Motion Planning for 3D Target Tracking among Obstacles. In *ISRR*, pages 267–279, 2007.

[Bares et al., 1998] William H. Bares, Joël P. Grégoire, and James C. Lester. Realtime Constraint-Based Cinematography for Complex Interactive 3D Worlds. In *AAAI/IAAI*, pages 1101–1106, 1998.

[Benhimane and Malis, 2006] Selim Benhimane and Ezio Malis. Homography-Based 2D Visual Servoing. In *ICRA*, pages 2397–2402, 2006.

[Bindel et al., 2002] David Bindel, James Demmel, William Kahan, and Osni Marques. On Computing Givens Rotations Reliably and Efficiently. *ACM Transactions on Mathematical Software*, 28(2):206–238, 2002.

[Bittner, 2002] Jiri Bittner. *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University, October 2002.

[Bradshaw and O'Sullivan, 2004] Gareth Bradshaw and Carol O'Sullivan. Adaptive Medial-Axis Approximation for Sphere-Tree Construction. *ACM Transactions on Graphics*, 23(1):1–26, 2004.

[Broberg, 2011] D.K. Broberg. Infrastructures for Home Delivery, Interfacing, Captioning, and Viewing of 3D Content. *Proceedings of the IEEE*, 99(4):684 –693, 2011.

[Byrne and Becker, 2008] Patrick Byrne and Suzanna Becker. A Principle for Learning Egocentric-Allocentric Transformation. *Neural Computation*, 20(3):709–737, 2008.

[Chan et al., 2005] H. P. Chan, M. M. Goodsitt, M. A. Helvie, L. M. Hadjiiski, J. T. Lydick, M. A. Roubidoux, J. E. Bailey, A. Nees, C. E. Blane, and B. Sahiner. ROC Study of the Effect of Stereoscopic Imaging on Assessment of Breast Lesions. *Medical Physics*, 32, 4:1001–1009, 2005.

[Chang et al., 2007] Youngha Chang, Suguru Saito, and Masayuki Nakajima. Example-Based Color Transformation of Image and Video Using Basic Color Categories. *IEEE Transactions on Image Processing*, 16(2):329–336, 2007.

[Chia et al., 2011] Alex Yong Sang Chia, Shaojie Zhuo, Raj Kumar Gupta, Yu-Wing Tai, Siu-Yeung Cho, Ping Tan, and Stephen Lin. Semantic Colorization with Internet Images. *ACM Transactions on Graphics*, 30(6):156, 2011.

[Christie and Olivier, 2009] Marc Christie and Patrick Olivier. Camera Control in Computer Graphics: Models, Techniques and Applications. In *SIGGRAPH ASIA Courses*, 2009.

[Christie et al., 2008] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera Control in Computer Graphics. *Comput. Graph. Forum*, 27(8):2197–2218, 2008.

[Cohen-Or et al., 2003] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva, and Frédo Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.

[Cohen, 2011] Noy Cohen. A Color Balancing Algorithm for Cameras. *EE368 Digital Image Processing*, 2011.

[Dale et al., 2009] Kevin Dale, Micah K. Johnson, Kalyan Sunkavalli, Wojciech Matusik, and Hanspeter Pfister. Image Restoration using Online Photo Collections. In *ICCV*, pages 2217–2224, 2009.

[David, 1963] H. A. David. *The Method of Paired Comparisons*. Charles Griffin & Company, 1963.

[Dechter and Pearl, 1985] Rina Dechter and Judea Pearl. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*, 32(3):505–536, 1985.

[Didyk et al., 2011] Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. A Perceptual Model for Disparity. *ACM Transactions on Graphics*, 30(4):96, 2011.

[Drucker and Zeltzer, 1994] Steven M. Drucker and David Zeltzer. Intelligent Camera Control in a Virtual Environment. In *Proceedings of Graphics Interface*, pages 190–199, 1994.

[Drummond and Cipolla, 2002] Tom Drummond and Roberto Cipolla. Real-Time Tracking of Complex Structures with On-Line Camera Calibration. *Image Vision Computation*, 20(5-6):427–433, 2002.

[Farbman and Lischinski, 2011] Zeev Farbman and Dani Lischinski. Tonal Stabilization of Video. *ACM Transactions on Graphics*, 30(4):89, 2011.

[Farbman et al., 2010] Zeev Farbman, Raanan Fattal, and Dani Lischinski. Diffusion Maps for Edge-Aware Image Editing. *ACM Transactions on Graphics*, 29(6):145, 2010.

[Farin, 1990] Gerald E. Farin. *Curves and Surfaces for Computer Aided Geometric Design - a Practical Guide*. Computer science and scientific computing. Academic Press, 1990.

[Fischler and Bolles, 1981] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting. *Commun. ACM*, 24(6):381–395, 1981.

[Fröhlich et al., 1999] Bernd Fröhlich, Stephen Barrass, Björn Zehner, John Plate, and Martin Göbel. Exploring Geo-Scientific Data in Virtual Environments. In *IEEE Visualization*, pages 169–173, 1999.

[Gateau and Neuman, 2010] Samuel Gateau and Robert Neuman. Stereoscopy From XY to Z. In *SIGGRAPH ASIA Courses*, 2010.

[Gleicher and Witkin, 1992] Michael Gleicher and Andrew P. Witkin. Through-the-Lens Camera Control. In *SIGGRAPH*, pages 331–340, 1992.

[Greß et al., 2006] Alexander Greß, Michael Guthe, and Reinhard Klein. GPU-Based Collision Detection for Deformable Parameterized Surfaces. *Computer Graphics Forum*, 25(3):497–506, 2006.

[Grinberg et al., 1994] V. S. Grinberg, Gregg Podnar, and Mel Siegel. Geometry of Binocular Imaging. In *Stereoscopic Displays and Virtual Reality Systems*, pages 56–65, 1994.

[Haigh-Hutchinson, 2009] Mark Haigh-Hutchinson. *Real-Time Cameras. A Guide for Game Designers and Developers.* Morgan Kaufmann, 2009.

[Halper and Masuch, 2003] Nick Halper and Maic Masuch. Action Summary for Computer Games. In *Proc. of 2nd International Conference on Application and Development of Computer Games*, pages 124–132, 2003.

[Halper et al., 2001] Nicolas Halper, Ralf Helbing, and Thomas Strothotte. A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence. *Comput. Graph. Forum*, 20(3):174–183, 2001.

[Heinzle et al., 2011] Simon Heinzle, Pierre Greisen, David Gallup, Christine Chen, Daniel Saner, Aljoscha Smolic, Andreas Burg, Wojciech Matusik, and Markus H. Gross. Computational Stereo Camera System with Programmable Control Loop. *ACM Transactions on Graphics*, 30(4):94, 2011.

[Held and Banks, 2008] Robert T. Held and Martin S. Banks. Misperceptions in Stereoscopic Displays: a Vision Science Perspective. In *APGV*, pages 23–32, 2008.

[Hertzmann et al., 2001] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340, 2001.

[Hoffman et al., 2008] David M. Hoffman, Ahna R. Girshick, Kurt Akeley, and Martin S. Banks. Vergence-Accommodation Conflicts Hinder Visual Performance and Cause Visual Fatigue. *Journal of Vision*, 8(3):1–30, 2008.

[Hornung et al., 2003] Alexander Hornung, Gerhard Lakemeyer, and Georg Trogemann. An Autonomous Real-Time Camera Agent for Interactive Narratives and Games. In *IVA*, pages 236–243, 2003.

[IEE, 2011] *Special Issue on 3D Media and Displays*, volume 99, 4. Proceedings of the IEEE, 2011.

[Ji et al., 2004] Ye Ji, Hong-Bo Liu, Xiu-Kun Wang, and Yi-Yuan Tang. *Color Transfer to Greyscale Images using Texture Spectrum*, volume 7, pages 4057–4061. 2004.

[Jones et al., 2001] Graham Jones, Delman Lee, Nicolas Holliman, and David Ezra. Controlling Perceived Depth in Stereoscopic Images. In *Stereoscopic Displays And Virtual Reality Systems VIII*, pages 200–1, 2001.

[Kagarlitsky et al., 2009] Sefy Kagarlitsky, Yael Moses, and Yacov Hel-Or. Piecewise-Consistent Color Mappings of Images Acquired Under Various Conditions. In *ICCV*, pages 2311–2318, 2009.

[Kang et al., 2010] Sing Bing Kang, Ashish Kapoor, and Dani Lischinski. Personalization of Image Enhancement. In *CVPR*, pages 1799–1806, 2010.

[Kennedy and Mercer, 2001] Kevin Kennedy and Robert E. Mercer. Planning Animations Using Cinematography Knowledge. In *Canadian Conference on AI*, pages 357–360, 2001.

[Kim et al., 2008] Hye Jin Kim, Jae Wan Choi, An-Jin Chaing, and Ki Yun Yu. Reconstruction of Stereoscopic Imagery for Visual Comfort. In *Stereoscopic Displays and Virtual Reality Systems XIV, SPIE Vol. 6803*, 2008.

[Klein and Murray, 2006] Georg Klein and David W. Murray. Full-3D Edge Tracking with a Particle Filter. In *BMVC*, pages 1119–1128, 2006.

[Klein and Murray, 2008] Georg Klein and David W. Murray. Improving the Agility of Keyframe-Based SLAM. In *ECCV (2)*, pages 802–815, 2008.

[Klein and Murray, 2010] Georg Klein and David W. Murray. Simulating Low-Cost Cameras for Augmented Reality Compositing. *IEEE Transactions on Vision and Computer Graphics*, 16(3):369–380, 2010.

[Knecht et al., 2011] Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive Camera-Based Color Mapping for Mixed-Reality Applications. In *ISMAR*, pages 165–168, 2011.

[Koppal et al., 2011] Sanjeev J. Koppal, C. Lawrence Zitnick, Michael F. Cohen, Sing Bing Kang, Bryan Ressler, and Alex Colburn. A Viewer-Centric Editor for 3D Movies. *IEEE Computer Graphics and Applications*, 31(1):20–35, 2011.

[Laine, 2005] Samuli Laine. A General Algorithm for Output-Sensitive Visibility Preprocessing. In *SI3D*, pages 31–40, 2005.

[Lang et al., 2010] Manuel Lang, Alexander Hornung, Oliver Wang, Steven Poulakos, Aljoscha Smolic, and Markus H. Gross. Nonlinear Disparity Mapping for Stereoscopic 3D. *ACM Transactions on Graphics*, 29(4), 2010.

[LaValle, 2006] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[Lazebnik, 2001] S. Lazebnik. Visibility-Based Pursuit Evasion in Three-Dimensional Environments. Technical Report CVR TR 2001-01, Beckman Institute, University of Illinois, 2001.

[Lee et al., 2010] Wonwoo Lee, Youngmin Park, Vincent Lepetit, and Woontack Woo. Point-and-Shoot for Ubiquitous Tagging on Mobile Phones. In *ISMAR*, pages 57–64, 2010.

[Lepetit and Fua, 2006] Vincent Lepetit and Pascal Fua. Keypoint Recognition Using Randomized Trees. *IEEE Transactions on Pattern Analanalysis*, 28(9):1465–1479, 2006.

[Levin et al., 2004] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using Optimization. *ACM Transactions on Graphics*, 23(3):689–694, 2004.

[Li and Cheng, 2008] Tsai-Yen Li and Chung-Chiang Cheng. Real-Time Camera Planning for Navigation in Virtual Environments. In *Smart Graphics*, pages 118–129, 2008.

[Li et al., 2010] Yong Li, Tao Ju, and Shi-Min Hu. Instant Propagation of Sparse Edits on Images and Videos. *Computer Graphics Forum*, 29(7):2049–2054, 2010.

[Liere and Mulder, 2003] Robert van Liere and Jurriaan D. Mulder. Optical Tracking Using Projective Invariant Marker Pattern Properties. In *Proceedings of the IEEE Virtual Reality 2003*, VR '03, pages 191–, Washington, DC, USA, 2003. IEEE Computer Society.

[Lipton, 1982] Lenny Lipton. *Foundations of the Stereoscopic Cinema: A Study in Depth*. Van Nostrand Reinhold Inc.,U.S., 1982.

[Lischinski et al., 2006] Dani Lischinski, Zeev Farbman, Matthew Uyttendaele, and Richard Szeliski. Interactive Local Adjustment of Tonal Values. *ACM Transactions on Graphics*, 25(3):646–653, 2006.

[Lowe, 2004] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Marchand and Courty, 2002] Éric Marchand and Nicolas Courty. Controlling a Camera in a Virtual Environment. *The Visual Computer*, 18(1):1–19, 2002.

[Masaoka et al., 2006] Kenichiro Masaoka, Atsuo Hanazato, Masaki Emoto, Hirokazu Yamanoue, Yuji Nojiri, and Fumio Okano. Spatial Distortion Prediction System for Stereoscopic Images. *Electronic Imaging*, 15(1), 2006.

[Masehian and Sedighizadeh, 2007] Ellips Masehian and Davoud Sedighizadeh. Classic and Heuristic Approaches in Robot Motion Planning–A Chronological Review. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 23, 2007.

[Maslennikova and Vezhnevets, 2007] Alla Maslennikova and Vladimir Vezhnevets. Interactive Local Color Transfer Between Images. *GraphiCon 2007*, 2007.

[Meesters et al., 2004] Lydia M. J. Meesters, Wijnand A. IJsselsteijn, and Piter J. H. Seuntiens. A Survey of Perceptual Evaluations and Requirements of Three-Dimensional TV. *IEEE Transactions on Circuits and Systems*, 14(3):381–391, 2004.

[Murrieta-Cid et al., 2004] Rafael Murrieta-Cid, Alejandro Sarmiento, Sourabh Bhattacharya, and Seth Hutchinson. Maintaining Visibility of a Moving Target at a Fixed Distance: the Case of Observer Bounded Speed. In *ICRA*, pages 479–484, 2004.

[Murrieta-Cid et al., 2007] Rafael Murrieta-Cid, Teja Muppirala, Alejandro Sarmiento, Sourabh Bhattacharya, and Seth Hutchinson. Surveillance Strategies for a Pursuer with Finite Sensor Range. *I. J. Robotic Res.*, 26(3):233–253, 2007.

[Neumann and Neumann, 2005] Attila Neumann and László Neumann. Color Style Transfer Techniques using Hue, Lightness and Saturation Histogram Matching. In *Computational Aesthetics*, pages 111–122, 2005.

[Niederberger et al., 2004] Christoph Niederberger, Dejan Radovic, and Markus H. Gross. Generic Path Planning for Real-Time Applications. In *Computer Graphics International*, pages 299–306, 2004.

[Oskam et al., 2009] Thomas Oskam, Robert W. Sumner, Nils Thürey, and Markus H. Gross. Visibility Transition Planning for Dynamic Camera Control. In *Symposium on Computer Animation*, pages 55–65, 2009.

[Oskam et al., 2011] Thomas Oskam, Alexander Hornung, Huw Bowles, Kenny Mitchell, and Markus Gross. OSCAM - Optimized Stereoscopic Camera Control for Interactive 3D. *ACM Transactions on Graphics*, pages 189:1–189:8, 2011.

[Oskam et al., 2012] Thomas Oskam, Alexander Hornung, Robert. W Sumner, and Markus Gross. Fast and Stable Color Balancing for Images and Augmented Reality. *Proceedings of 3DIMPVT*, 2012.

[Oskam, 2008] Thomas Oskam. Visibility Transition Planning For Real-Time Camera Control. Master's thesis, Eidgenössische Technische Hochschule (ETH) Zürich, Switzerland, 2008.

[Pan et al., 2011] Hao Pan, Chang Yuan, and Scott Daly. 3D Video Disparity Scaling for Preference and Prevention of Discomfort. In *Stereoscopic Displays and Applications XXII, SPIE Vol. 7863*, 2011.

[Park et al., 2009] Youngmin Park, Vincent Lepetit, and Woontack Woo. ESM-Blur: Handling & Rendering Blur in 3D Tracking and Augmentation. In *ISMAR*, pages 163–166, 2009.

[Petersen and Pedersen, 2008] Kaare Brandt Petersen and Michael Syskind Pedersen. The Matrix Cookbook. 2008.

[Pham and Pringle, 1995] Binh Pham and Glen Pringle. Color Correction for an Image Sequence. *IEEE Computer Graphics Applications*, 15(3):38–42, May 1995.

[Pitié and Kokaram, 2007] François Pitié and Anil C. Kokaram. The Linear Monge-Kantorovitch Linear Colour Mapping for Example-Based Colour Transfer. *Visual Media Production*, 2007.

[Pitié et al., 2007] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. Automated Colour Grading using Colour Distribution Transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, 2007.

[Porikli, 2003] Fatih Murat Porikli. Inter-Camera Color Calibration by Correlation Model Function. In *ICIP (2)*, pages 133–136, 2003.

[Pouli and Reinhard, 2011] Tania Pouli and Erik Reinhard. Progressive Color Transfer for Images of Arbitrary Dynamic Range. *Computers & Graphics*, 35(1):67–80, 2011.

[Reinhard et al., 2001] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color Transfer Between Images. *IEEE Computer Graphics and Applications*, 21(5):34–41, 2001.

[Salomon et al., 2003] Brian Salomon, Maxim Garber, Ming C. Lin, and Dinesh Manocha. Interactive Navigation in Complex Environments using Path Planning. In *SI3D*, pages 41–50, 2003.

[Senanayake and Alexander, 2007] C. R. Senanayake and Daniel C. Alexander. Colour Transfer by Feature Based Histogram Registration. In *BMVC*, 2007.

[Shepard, 1968] Donald Shepard. A Two-Dimensional Interpolation Function for Irregularly-Spaced Data. In *Proceedings of the 1968 23rd ACM national conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM.

[Shibata et al., 2011a] Takashi Shibata, Joohwan Kim, David M. Hoffman, and Martin S. Banks. The Zone of Comfort: Predicting Visual Discomfort with Stereo Displays. *Journal of Vision*, 11(8), 2011.

[Shibata et al., 2011b] Takashi Shibata, Joohwan Kim, David M. Hoffman, and Martin S. Banks. Visual Discomfort with Stereo Displays: Effects of Viewing Distance and Direction of Vergence-Accommodation Conflict. In *Stereoscopic Displays and Applications XXII, SPIE Vol. 7863*, 2011.

[Shoemake, 1985] Ken Shoemake. Animating Rotation with Quaternion Curves. In *SIGGRAPH*, pages 245–254, 1985.

[Siddiqui and Bouman, 2008] Hasib Siddiqui and Charles A. Bouman. Hierarchical Color Correction for Camera Cell Phone Images. *IEEE Transactions on Image Processing*, 17(11):2138–2155, 2008.

[Siméon et al., 2000] Thierry Siméon, Jean-Paul Laumond, and Carole Nissoux. Visibility-Based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics*, 14(6):477–493, 2000.

[Smolic et al., 2011] A. Smolic, P. Kauff, S. Knorr, A. Hornung, M. Kunter, M. Müller, and M. Lang. Three-Dimensional Video Postproduction and Processing. *Proceedings of the IEEE*, 99(4):607–625, 2011.

[Stelmach et al., 2003] Lew B. Stelmach, Wa James Tam, Filippo Speranza, Ronald Renaud, and Taali Martin. Improving the Visual Comfort of Stereoscopic Images. In *Proc. SPIE 5006, 269*, 2003.

[Tai et al., 2005] Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang. Local Color Transfer via Probabilistic Segmentation by Expectation-Maximization. In *CVPR (1)*, pages 747–754, 2005.

[Tian et al., 2002] Gui Yun Tian, Duke Gledhill, David Taylor, and David Clarke. Colour Correction for Panoramic Imaging. In *IV*, pages 483–488, 2002.

[Tomlinson et al., 2000] Bill Tomlinson, Bruce Blumberg, and Delphine Nain. Expressive Autonomous Cinematography for Interactive Virtual Environments. In *Agents*, pages 317–324, 2000.

[van de Panne and Stewart, 1999] Michiel van de Panne and A. James Stewart. Effective Compression Techniques for Precomputed Visibility. In *Rendering Techniques*, pages 305–316, 1999.

[Varadhan and Manocha, 2005] Gokul Varadhan and Dinesh Manocha. Star-shaped Roadmaps - A Deterministic Sampling Approach for Complete Motion Planning. In *Robotics: Science and Systems*, pages 25–32, 2005.

[Vidal et al., 2002] René Vidal, Omid Shakernia, H. Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation. *IEEE Transactions on Robotics*, 18(5):662–669, 2002.

[Wang and Sawchuk, 2008] Chiao Wang and Alexander A. Sawchuk. Disparity Manipulation for Stereo Images and Video. In *Stereoscopic Displays and Applications XIX, SPIE Vol. 6803*, 2008.

[Wang et al., 2010] Baoyuan Wang, Yizhou Yu, Tien-Tsin Wong, Chun Chen, and Ying-Qing Xu. Data-Driven Image Color Theme Enhancement. *ACM Transactions on Graphics*, 29(6):146, 2010.

[Wang et al., 2011] Baoyuan Wang, Yizhou Yu, and Ying-Qing Xu. Example-Based Image Color and Tone Style Enhancement. *ACM Transactions on Graphics*, 30(4):64, 2011.

[Watt et al., 2005] Simon J. Watt, Kurt Akeley, Marc O. Ernst, and Martin S. Banks. Focus Cues Affect Perceived Depth. *Journal of Vision*, 5(10), 2005.

[wei He et al., 1996] Li wei He, Michael F. Cohen, and David Salesin. The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing. In *SIGGRAPH*, pages 217–224, 1996.

[Welsh et al., 2002] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring Color to Greyscale Images. In *SIGGRAPH*, pages 277–280, 2002.

[Wen et al., 2008] Chung-Lin Wen, Chang-Hsi Hsieh, Bing-Yu Chen, and Ming Ouhyoung. Example-Based Multiple Local Color Transfer by Strokes. *Computer Graphics Forum*, 27(7):1765–1772, 2008.

[Woods et al., 1993] Andrew Woods, Tom Docherty, and Rolf Koch. Image Distortions in Stereoscopic Video Systems. In *Stereoscopic Displays and Applications IV, Proceedings of the SPIE*, volume 1915, 1993.

[Wyszecki and Stiles, 1967] Günther Wyszecki and Walter Stanley Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulas*. Wiley and Sons, Inc. New York, 1967.

[Xiao and Ma, 2006] Xuezhong Xiao and Lizhuang Ma. Color Transfer in Correlated Color Space. In *VRCIA*, pages 305–309, 2006.

[Xiao and Ma, 2009] XueZhong Xiao and Lizhuang Ma. Gradient-Preserving Color Transfer. *Computer Graphics Forum*, 28(7):1879–1886, 2009.

[Xiong and Pulli, 2010] Yingen Xiong and Kari Pulli. Color and Luminance Compensation for Mobile Panorama Construction. In *ACM Multimedia*, pages 261–270, 2010.

[Yang and LaValle, 2002] Libo Yang and Steven M. LaValle. An Improved Random Neighborhood Graph Approach. In *ICRA*, pages 254–259, 2002.

[Yang et al., 2011] Sejung Yang, Yoon-Ah Kim, Chaerin Kang, and Byung-Uk Lee. Color Compensation Using Nonlinear Luminance-RGB Component Curve of a Camera. In *ISVC (2)*, pages 617–626, 2011.

[Zilly et al., 2011] F. Zilly, J. Kluger, and P. Kauff. Production Rules for Stereo Acquisition. *Proceedings of the IEEE*, 99(4):590–606, 2011.