Internship Thesis

Eastern Perspective - Multi Projection Images

Lothar Schlesier
Faculty of Computer Science
Department of Simulation and Graphics
Otto-von-Guericke University Magdeburg
lschlesi@cs.uni-magdeburg.de

31st May 2004

Supervisors:
Prof. Dr. Sheelagh Carpendale,
Prof. Dr. Thomas Strothotte,
Henry Sonnet

Schlesier, Lothar:
Eastern Perspective - Multi Projection Images
Internship Thesis, Otto-von-Guericke University Magdeburg, 2004.

Supervisors:

Prof. Dr. Sheelagh Carpendale
University of Calgary (Department of Computer Science)
Prof. Dr. Thomas Strothotte
Otto-von-Guericke University (Department of Simulation and Graphics)
Henry Sonnet
Otto-von-Guericke University (Department of Simulation and Graphics)

Location:

Department of Computer Science
Innovations in Visualizations Laboratory
University of Calgary
2500 University Drive N.W.
Calgary Alberta
Canada T2N 1N4

Acknowledgement

I would like to thank Prof. Dr. Sheelagh Carpendale, Prof. Dr. Thomas Strothotte, and Henry Sonnet for supervising and advising me during the internship. I also would like to thank all members of the ILab, the GroupLab and the Graphics Jungle for their ideas and support and for the great time we had at work as well as in our spare time. You made the time of the internship a very special and important experience for me. Thanks are also going to my family and friends who had understanding for the time we were separated and who kept motivating me throughout the internship and the writing of this thesis.

Contents

List of Figures **vi**

List of Abbreviations **vii**

1 Introduction **1**

2 Fine Arts and Computer Graphics Fundamentals **3**

2.1 Chinese Landscape Painting 3

2.1.1 Impression of Space 3

2.1.2 Depth 4

2.1.3 Structure of Chinese Landscape Painting 5

2.1.4 Depiction of Man-made Things 6

2.1.5 Comparison of an Eastern and a Western Example 8

2.2 Computer Graphics Fundamentals 9

2.2.1 The Rendering Pipeline 9

2.2.2 Planar Geometric Projections 10

3 Related Work **19**

3.1 Multiple Projections 19

3.2 Eastern Perspective 20

3.3 Simulation of Chinese Drawing Styles 22

4 Requirements and Conception **23**

4.1 Basic Requirements 23

4.2 A Rendering Pipeline for Multi Projection Images 24
4.3 Image Composition 25
4.4 Projection Modifiers 27
4.4.1 Post Projection Shifts 27
4.4.2 Dependencies between Scene Parts 28
4.5 Picking in Multi Projection Images 30
4.6 Object Stacking 31
4.7 A Concept for a User Interface 33

5 Implementation **35**
5.1 Resources 35
5.2 Base Classes 36
5.2.1 3D Input 36
5.2.2 Scene Part 37
5.2.3 Projections 37
5.2.4 QExtNumberEdit 38
5.3 Main Window 38
5.4 Scene Part Settings Window 40
5.5 Scene View Window 42
5.6 Result Images 44

6 Conclusion and Future Work **49**

Bibliography **51**

List of Figures

2.1 Deep distance 5
2.2 Level distance and high distance 6
2.3 Depiction of man-made things 6
2.4 Depiction of man-made things 7
2.5 *Landscape* by *Tang Yin* 8
2.6 *Entrance to the village of Voisins* by *Camille Pissarro* 9
2.7 Traditional rendering pipeline 10
2.8 Perspective projection components 11
2.9 Vanishing points 13
2.10 Parallel projection components 14
2.11 Example for orthographic projection 15
2.12 Example for a trimetric projection 16
2.13 Example for a oblique projection 17

3.1 Multi projection image 19
3.2 Non-linear perspective 20
3.3 Example for a stacking projection 21
3.4 Converting photos into drawings 21
3.5 Brush stroke texture for mountains 22
3.6 Brush stroke texture for a tree 22

4.1 MPI rendering pipeline 24
4.2 MPI composition 26

4.3 Post projection shifts, motivation 27
4.4 Post projection shifts, solution 28
4.5 Scene part dependencies 29
4.6 Limitations of scene part dependencies 30
4.7 MPI picking data structure correlation 31
4.8 Object stacking 32
4.9 Component layout 34

5.1 Main Window 39
5.2 File dialog 40
5.3 Scene Part Settings Window 41
5.4 Scene View Window 43
5.5 Example 1, parallel projected 44
5.6 Example 1, MPI 45
5.7 Example 2, single perspective projection. 46
5.8 Example 2, MPI, oblique projections 46
5.9 Example 2, MPI, different oblique projections 46
5.10 Example 3, stacking projection and object stacking 47
5.11 Example 3, object stacking algorithm problems 48

List of Abbreviations

2D	two dimensional
3D	three dimensional
AI	Artificial Intelligence
API	Application Programming Interface
CG	Computer Graphics
COP	Center of Projection
CSCW	Computer Supported Cooperative Work
DOP	Direction of Projection
GUI	Graphical User Interface
HCI	Human-Computer-Interaction
HSR	Hidden Surface Removal
ILab	Innovations in Visualizations Laboratory
MDI	Multi Document Interface
MPI	Multi Projection Image
NPR	Non-photo realistic rendering
P	Point
P'	Projected Point of P
P_{box}	Orthographic Projection (with a box-shaped viewing volume)
P_{fr}	Perspective Projection (with a frustum-shaped viewing volume)
P_{ortho}	Orthographic Projection
$P_{oblique}$	Oblique Projection
P_{persp}	Perspective Projection

Chapter 1

Introduction

The basis for this thesis is an internship at the Innovations in Visualizations Laboratory of the Department of Computer Science of the Faculty of Science at the University of Calgary in Calgary, Canada. An internship is part of the diploma degree program in Computer Science at the Otto-von-Guericke University in Magdeburg, Germany. The internship lasted from April 3rd, 2003 until September 30th, 2003.

The ILab is headed by Prof. Dr. Sheelagh Carpendale. Several PhD- and graduate students as well as interns work on tasks which involve interactive spatial organization, information visualization and interface design. The ILab is associated with the Laboratory for HCI and CSCW (GroupLab) headed by Prof. Dr. Saul Greenberg. The GroupLab is specialized in Human-Computer-Interaction and Computer Supported Cooperative Work. A great interexchange of ideas and discussions take place between both laboratories, resulting in a lot of creativity and great atmosphere to work in.

The task for the internship was to implement a software that allows a user to create multi projection images from a given 3D scene. Different types of planar geometric projections shall be integrated and interaction methods for working and adjusting these projections shall be provided. The inspiration for this work are Chinese Landscape Paintings with their multiple projections and viewpoints in a single image. The artists used this technique to create a feeling of endless space and also to show the important aspects of things. This shows that a single (perspective) projection as it is often used in western art as well as in Computer Graphics is not suitable for every application. This work is a step to overcome the limitation of a single projection in CG. It deals only with the geometric aspects of multi projection images and not with the simulation of Chinese drawing styles, brushes and ink.

As *Agrawala* et al. point out in [AZM00] there are several applications for multi projection images:

- *Artistic Expression*: multiple projections allow artists to express a certain mood, feeling or idea. Chinese Landscape Paintings often create a feeling of an unlimited space, that is because of the use of different projections and viewpoints. For these

images the viewer could get the feeling of wandering through the scene and not only watching it like through a window.

- *Representation*: multiple projections can improve the representation or comprehensibility of a scene. Certain viewpoints are better than others for comprehending the 3D nature of an object. Choosing different viewpoints for different objects allows to display the most interesting aspects of each individually. Also for a large formated image a single projection is often inadequate. For wide angle perspective projections, objects close to the image plane and close to the image borders can appear quite distorted. To solve this multiple projections are suitable again.

- *Visualization*: different projections can be used as another cue beside position, size, color, lighting and others to differentiate between objects or to lead the focus of the viewer.

The result of the internship is a program called MPI-Builder which implements a rendering pipeline for the creation of multi projection images.

Structure

The remainder of the thesis is structured as follows: Chapter 2 argues the fundamentals for this work. These are Chinese Landscape Paintings and especially their composition on the one hand and CG basics such as planar geometric projections on the other hand. Chapter 3 takes a look on related work that deals with the creation of multi projection images, the simulation of eastern perspective, and the simulation of Chinese drawing styles. A rendering pipeline for multi projection images is introduced in Chapter 4. Furthermore requirements and concepts for an application are discussed. Chapter 5 describes the implementation of the MPI-Builder and presents result images created with that application. A conclusion and a look onto further development is given in Chapter 6.

Chapter 2

Fine Arts and Computer Graphics Fundamentals

The goal of this chapter is to expose the fundamentals for this work. Chinese Landscape Paintings and especially their composition are its inspiration. They are discussed in the first part. The traditional computer graphics rendering process and planar geometric projections that are used in CG are the topic of the second part.

2.1 Chinese Landscape Painting

Chinese Landscape Painting refers to a style of eastern art that started to develop in the fifth century [Cah77] and that is popular till nowadays. As the Chinese words for this style *shan shui*, which means *mountain* and *water*, indicate important elements are hills, mountains, rivers and lakes. Further elements are architecture and humans in their environment. Although artists used various materials for the creation of their images so that different drawing styles can be found there are concepts for the composition that are quite common for all of them. Generally it can be stated that Chinese Landscape Paintings are multi projection images, as different viewpoints and even different projections are used in different parts of an image. This is also referred to as *Eastern Perspective* [1].

2.1.1 Impression of Space

For a long time the goal of western painters was to create realistic images. This results in the development and use of perspective projections in the Renaissance. Their images often were meant to be placed on a wall to serve like a window. That fact, the vanishing points (caused by the perspective projections) that strongly lead the focus of the viewer, and the strongly defined borders of the image give the impression of entering the displayed

[1] Throughout this thesis the expression perspective projection is explicitly used to describe this special type of projection. In fine arts the word perspective is not only used in the geometric sense.

space. Chinese Landscape Paintings are created in a different attitude: the artists wanted to create the impression of infinite space opening up in front of the viewer [HH91]. By viewing an image a person should be able to mentally walk through the landscape and see different things from different positions. So the painters overcame the limitation of a fixed viewpoint and chose different viewpoints for different parts of the image to display the important aspects of things. The following poem *Mount Lu* (taken from [dS68]) by *Su Tung-p'o* nicely illustrates this idea:

From that side it seems a peak
From this side a range,
View it from on high, view it from below
It is never twice the same -
How comes it that we cannot know
This mountain's real form?
It is, oh friend, it is that we
Are Dwellers on the Mount of Lu.

In Chinese Landscape Paintings the horizon is not decisive. As David Hockney points out in [HH91] leaving empty areas in an image that can be filled by the imagination of the viewer enhances the impression of infinite space, whereas space that is completely filled up with objects appears limited.

Very long or high scrolls were a typical format used by the artists. On the one hand this is another contrast to western art: while the western images have fixed borders, different parts of the scroll can be set into focus by unrolling it. On the other hand this extreme format does not allow the use of one single perspective projection as this would cause a lot of distortions at the distant sides. That is why parallel projections were used quite often. This results in an enhancement of the impression of endless space as parallel lines do not intersect in distance as it might be expected.

2.1.2 Depth

A general problem in painting of real world things is to express the 3D character of objects on a 2D surface. In Chinese Landscape Paintings the following concepts can be found: occlusion, level of detail, atmospheric perspective and horizontal position.

- *Occlusion* or *Interposition* is a depth cue based on the partial obscuration of a distant object by a closer one so that the obscured object is perceived as the one further away. Often mist is used to obscure objects referred to as misty depth. The use of mist enhances the feeling of endless space as distances can hardly be estimated. The illusion of height is created by covering the bases of mountains with mist.

- The *position* or distance of an object to the artist influences its position in the image: the further away the object is, the further up it is painted.
- *Level of detail*: the closer an object is the more details are displayed. The following poem (taken from [dS68]) by *Wang Wei* conveys this idea:

  In the distance men have no eyes,
  Trees have no branches, mountains
  No stones and water no waves.

- *Atmospheric perspective* is a method that creates an illusion of space and distance: distant objects are seen less clearly than those that are closer because of the scattering of light that is caused by small dust particles or water suspended into the air. This effect is created by displaying objects in progressively lighter tones as they get further away [Cah77].

2.1.3 Structure of Chinese Landscape Painting

The image space in a Chinese Landscape Painting is divided into three spaces separated by horizontal planes: foreground, middle part, background. Different viewpoints for these three parts are quite common. Furthermore different viewpoints can be found in one space itself again. This results in several layers the image is composed of.

Three general concepts are known that can be directly applied to the three spaces [dS68].

- ***Deep distance*** or *shen-yüan* is like standing at the foot of a mountain and looking up to its top (see Figure 2.1). This makes the mountain appear more towering.
- ***Level distance*** or *p'ing-yüan* is like standing on the ground and viewing into the distance. This is shown in the left part of Figure 2.2.
- ***High distance*** or *kao-yüan* is like looking from a high position such as a mountain down onto the landscape (see right part of Figure 2.2). This view gives a sense for the extension of areas. Objects that would be covered by closer ones can become partially visible. At its extreme, this view can become a birds eye view so that objects are seen from a position nearly perpendicular above them.

Figure 2.1: An example of *deep distance*. After a painting ascribed to *Hsü Tao-ning*. Taken from [dS68].

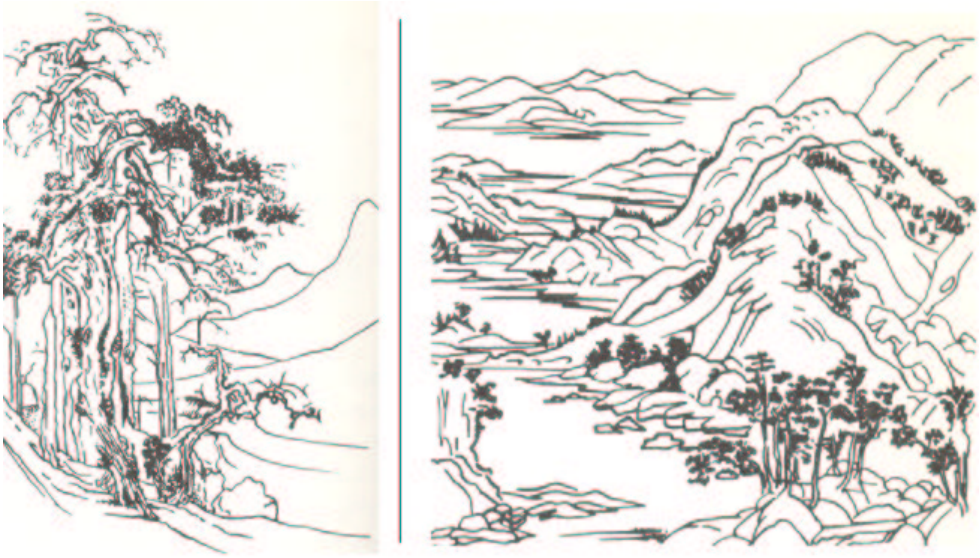

Figure 2.2: An example of *level distance*. After a painting ascribed to *Kuo Hsi* (left). And an example of *high distance*. After a painting ascribed to *Tung Yüan* (right). Taken from [dS68].

2.1.4 Depiction of Man-made Things

Man-made things such as architecture, roads, boats, furniture, etc. are often displayed by using an oblique projection. This fits into the idea of things that are further away, are painted further up. Multiple viewpoints are achieved by using different angles for the slope of lines of different objects. An example for this can be seen in Figure 2.3: the pools and mats in the foreground are seen from a position somewhere right of them, as their right sides are turned towards the viewer. The mat in the mid part is seen from a position more left as its left border is turned towards the viewer.

Another example is given in Figure 2.4. The palace, the furniture inside it, the street, the fences, and the cart are all displayed using oblique projections.

As already mentioned the fact that parallel lines of an object remain parallel in the image the impression of an infinite space is enhanced.

A definition and an explanation of the oblique projection can be found in the second part of this chapter in section 2.2.2.

Figure 2.3: Part of a cave drawing. It shows the use of different oblique projections. Taken from [dS68].

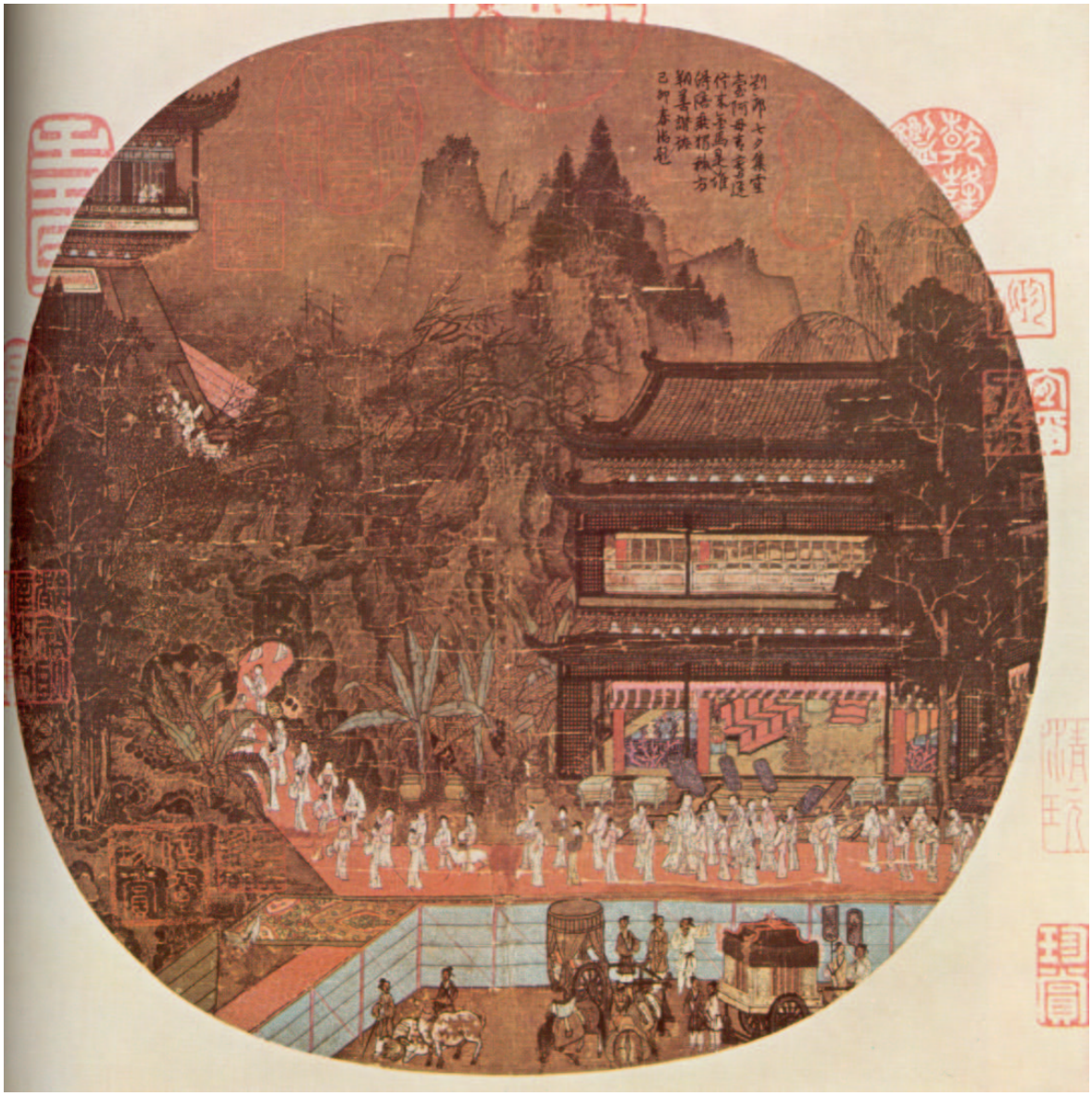

Figure 2.4: *The Han Palace* by an anonymous painter. Many examples for the use of oblique projections can be found in this painting. Taken from [Cah77].

2.1.5 Comparison of an Eastern and a Western Example

The scroll *Landscape* painted by *Tang Yin* shown in Figure 2.5 is a typical composition of a Chinese Landscape Painting that uses a lot of the previously discussed elements. The foreground is seen as standing on the ground (*level distance*). The mid part is seen from a higher position (*high distance*) and an oblique projection is used to display the pavilion. The mountains in the background, on the other hand, are seen from a lower position. This makes them appear towering (*deep distance*). Because of the use of different viewpoints there is no fixed predetermined point a viewer might focus. The way the picture is composed makes it easier for a viewer to imagine walking through the displayed landscape and looking around.

Figure 2.5: *Landscape* by *Tang Yin* as an example for eastern perspective. Taken from [Chi04].

The space between the mid part and the background is covered by mist (misty depth). The distance between those two parts cannot be estimated. This creates a feeling of infinite space. Closer objects are painted more detailed than those further away (level of detail). The mountains that are closer to the viewer have some detail. The peaks behind them are painted in lighter colors the further away they are (areal perspective). Only their shapes remain visible, no further details are shown. Eventually, this creates an impression of an endless space opening up in front of the viewer.

In contrast the painting *Entrance to the village of Voisins* by *Camille Pissarro* in the impressionistic style is an example for a typical western image composition. As shown in Figure 2.6 it gives the impression of entering the space. This is because of the use of a perspective projection. Lines going into the distance will intersect in a vanishing point. A good example is the way leading from the foreground into the background. This leads the focus of the viewer.

Because of the use of multiple viewpoints in the Chinese scroll relatively more image space is covered by ground than in Pissarro's painting.

After this comparison it should be obvious that the use of multiple viewpoints and projections can overcome the limitations of a single projection. With multiple viewpoints and the projection of the interesting parts of scenes, objects can be shown more emphasized.

Figure 2.6: *Entrance to the village of Voisins* by *Camille Pissarro* as an example for western perspective. Taken from [Sta94].

2.2 Computer Graphics Fundamentals

Computer Graphics is the field of Computer Science that deals with the creation and manipulation of images, the ways of displaying them, soft- and hardware that is needed to do this and much more. In this section the fundamentals for this work are discussed. It starts with a brief look on the process of creating an image from 3D data and ends up comparing different kinds of planar geometric projections.

2.2.1 The Rendering Pipeline

The process of creating an image from some 3D model data involves different steps that are executed one after the other. This sequence is often referred to as rendering pipeline. A basic rendering pipeline is shown in Figure 2.7. Its steps are the following:

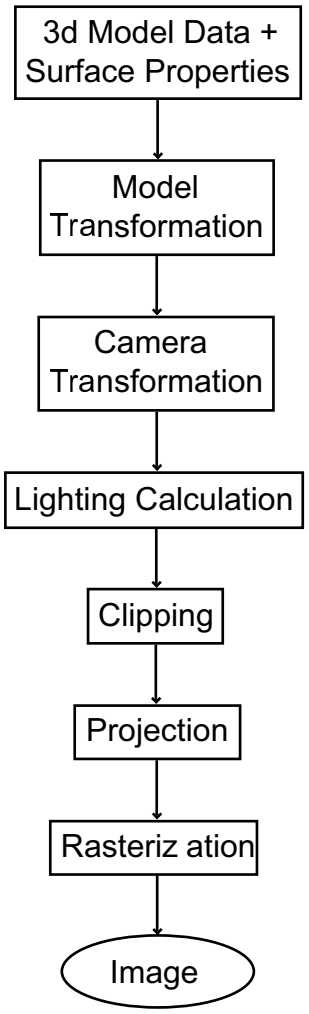

Figure 2.7: A simple six stage rendering pipeline.

The input into the rendering process is the model data including object geometry and object surface properties such as material and texture.

The geometry data can be modified by a series of transformations involving translations, rotations, scalings, and shearings.

The camera is positioned and oriented by a series of translations and rotations. The combination of model transformations with camera transformations determines the final relative position and orientation of the model to the camera.

The objects' colors are computed using their surface properties and the light properties.

Optionally all non visible objects can be removed to reduce rendering costs in the following steps. This might be done by testing the objects against the viewing volume.

The projection transforms the model from the 3D space into a 2D plane.

Finally the projected objects are drawn pixel by pixel into the framebuffer using the computed color information. E.g. z-buffer information can be used to determine visibility of objects in the resulting image.

The description above is very simplified. There are several modifications and extensions to achieve different results and to improve the performance of the rendering process. A more detailed description of the single steps in the rendering pipeline can be found in [FvDFH90].

In the case of a 3D space, the previously mentioned transformations as well as the projections are represented by 4×4 matrices. This is to compute the combination of any types of transformations by only using matrix multiplications. The resulting transformation matrix is then multiplied to the vectors representing the vertices of the model.

The next section takes a closer look on planar geometric projections, a very important step in the rendering pipeline for this work.

2.2.2 Planar Geometric Projections

Analogical to a person using a camera to take pictures of the real world it is possible to think of a virtual camera taking pictures in the virtual 3D world. The virtual camera has a position and orientation in the 3D world and uses a projection to create an image of a specific size. The position and orientation can be modified by the camera transformation.

A projection is a mapping from a higher dimensional space into a lower dimensional space. It is defined by a projection space and a set of projectors from this space into the world. Every point that is located on a specific projector is mapped onto the projector's starting point (see Figure 2.8). A planar geometric projection is a projection from a 3D into a 2D space. The projection space is a plane and the projectors are straight lines.

The projection used by the camera is specified by a set of parameters depending on its type. The projection plane is located at a fixed position relative to the camera. This means, if the camera is transformed the position and orientation of the projection plane change as well.

As the size of the image to be created is limited by the output device (e.g. screen resolution) only a small rectangular region in the projection plane is actually used. The projectors that intersect the rectangle define a viewing volume. Each object that is located inside the viewing volume is potentially visible in the resulting image.

As given by the above definition and illustrated in Figures 2.8 and 2.10, two different points that are intersected by the same projector are mapped onto the same point in the projection plane. While mathematical correct it might become a problem in CG: consider two (solid, nontransparent) objects are projected onto the same area in the projection plane. The object closer to the viewer shall be displayed. As the rendering process is a sequence, it can happen, that the object further away from the viewer is rendered last, overwriting the projection of the closer object. To eliminate this problems, techniques known as hidden surface removal can be used. A detailed discussion on HSR can be found in [FvDFH90].

Perspective Projections

Traditionally planar geometric projections are grouped into two classes depending on the directions of the projectors.

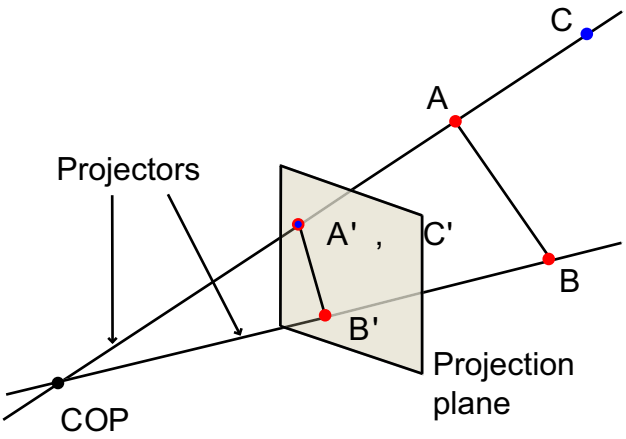

Figure 2.8: Perspective projection: determined by the center of projection (COP) in which the projectors intersect.

Perspective projections are planar geometric projections where all the projectors intersect in one point, the so called center of projection (see Figure 2.8). The basis for the calculation of the projected point $P'(x', y', z')$ of a point $P(x, y, z)$ is the similarity of triangles. Given z' as the distance from the COP to the projection plane, then x' and y' computes as follows: $x' = \frac{x}{z} * z'$ and $y' = \frac{y}{z} * z'$. Therefore the size of the projection of an object depends on the distance of that object to the COP and the distance of the projection plane to the COP. The further an object is away from the COP the smaller its projection is. If the projection plane is between the object and the COP, the projection of the object is smaller than the object. If the object is between the projection plane and the COP, the projection is bigger than the object.

The first case results in a realistic view like a photography and reveals the 3D nature of an object. That is why perspective projections are used for presentation drawings of architecture, industrial design and engineering as well as in a computer games. On the other hand it is not possible to take accurate measurements.

Usually the COP is located at a centered position behind the actual used rectangle of the projection plane. If it is moved out of the center in a specific direction points that are further away appear projected further in that direction. E.g. if the COP is moved up, points that are further away are projected further up. Projections with the COP moved up or down are discussed in [MCMW01] as *recession* projection.

A property of perspective projections is the existence of one or more vanishing points. A vanishing point is located on an axis of the object's coordinate system. Lines that are parallel to the axis intersect in that point. There can be only one vanishing point for each axis, and the axis needs to intersect the projection plane. Depending on the number of vanishing points a perspective projection is also called one-point, two-point, or three-point perspective. These three types are illustrated in Figure 2.9.

The viewing volume for perspective projections is a (infinite) pyramid. If the COP is not at a centered position the pyramid is oblique. To introduce a field of view (e.g., to avoid objects that are located between the projection plane and the COP), this volume can be truncated resulting in a frustum. The projection matrix for the perspective projection is defined by

$$P_{persp} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & s_z/d & 1 \end{bmatrix}$$

s_x, s_y, s_z are the scaling factors for the x, y, z components of a vertex, and d is the distance between the projection plane and the COP. P is defined as long as $d \neq 0$.

The projection matrix for the perspective projection with a truncated viewing volume is defined by

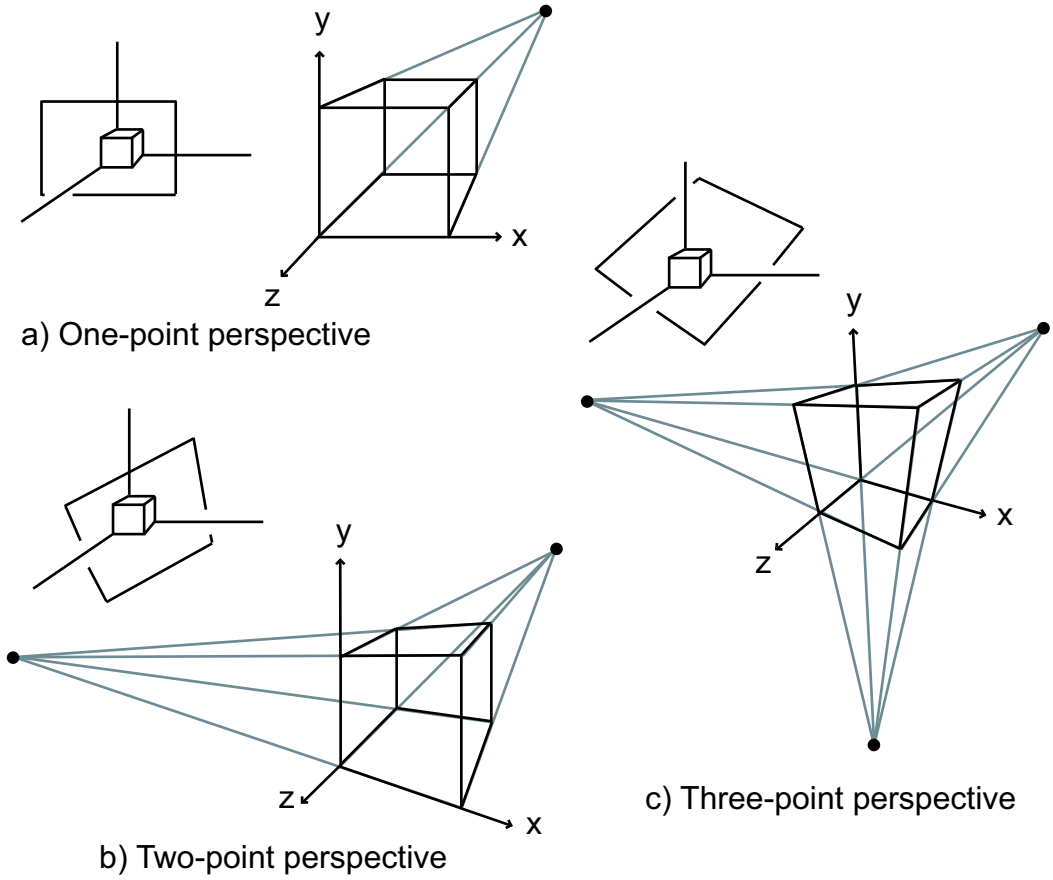

Figure 2.9: One, two, or three vanishing points, depending on the number of axes intersecting the projection plane.

$$P_{fr} = \begin{bmatrix} 2*N/(R-L) & 0 & (R+L)/(R-L) & 0 \\ 0 & 2*N/(T-B) & (T+B)/(T-B) & 0 \\ 0 & 0 & -(F+N)/(F-N) & -2*F*N/(F-N) \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

L is the shortest distance of the viewing volume's left plane to the z-axis of the camera's coordinate system, R is the shortest distance of the viewing volume's right plane to that z-axis, B is the shortest distance of the viewing volume's bottom plane to the camera's z-axis, T is the shortest distance of the viewing volume's top plane to the camera's z-axis, N is the distance of the viewing volume's front plane to the camera, and F is the distance of the viewing volume's back plane to the camera. P_{fr} is defined as long as $L \neq R$, $B \neq T$ and $N \neq F$.

Parallel Projections

Parallel projections are planar geometric projections where all the projectors are parallel to each other. The direction of projection or the angle between the projectors and the projection plane leads to further subclassing. All parallel projections have the following properties in common: angles between lines that are located on a plane that is parallel to the projection plane remain in the projected image; the projection of any pair of parallel lines is also parallel; parallel lines not parallel to the projection plane are equally foreshortened; as long as no scaling is applied to the projection, the distance between two points on a plane parallel to the projection plane is the same as the distance between the projection of those points.

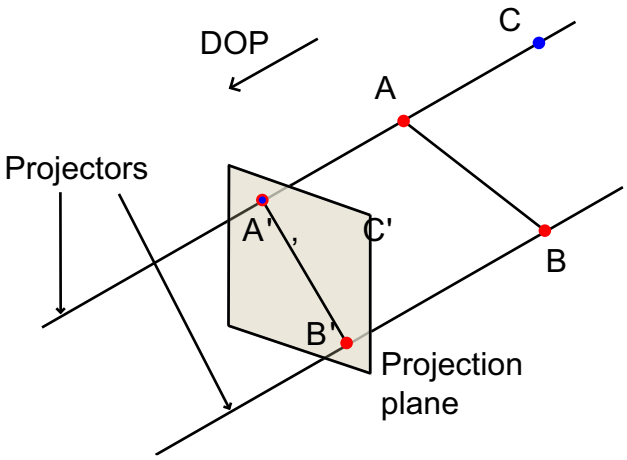

Figure 2.10: Parallel projection: determined by direction of projection (DOP). Projectors are parallel to each other.

Orthographic Projection

The orthographic projection is a parallel projection with the projectors perpendicular to the projection plane. Predicting a coordinate system with its origin in the projection plane and its z-axis perpendicular to the plane (the projectors are parallel to the z-axis), the projection of a point in this coordinate system results from removing the z-coordinate of that point.

The properties of parallel projections allow accurate measurements. Therefore the orthographic projection is used for engineering drawings of products (like machine parts, furniture, etc.) and architectural drawings. Especially for the first application a multi projection is common. This multi projection provides three separated views of the model, showing its front, side and top surface. As shown in Figure 2.11 many 3D modeling software tools like Maya also provide this option. The user can manipulate the scene in each of those views. Information regarding positions can be read more accurately from parallel projected views than from perspective views.

On the other hand the orthographic projection does not provide a realistic view, especially depth information gets lost. It does not provide a sense of the 3D character of an object. The viewing volume of this projection is a box with infinite extension in positive and negative z-direction. There is also a truncated version with clipping planes making it a closed volume, defining a field of view.

The projection matrix for the orthographic projection is defined by

$$P_{ortho} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with s_x, s_y as the scaling factors for the x, y components of a vertex. A value s_z for scaling the appropriate z coordinate is not needed, as the z component is lost during the projection. Side, top or any other views of the scene can be achieved by changing the camera's relative position and orientation towards the model.

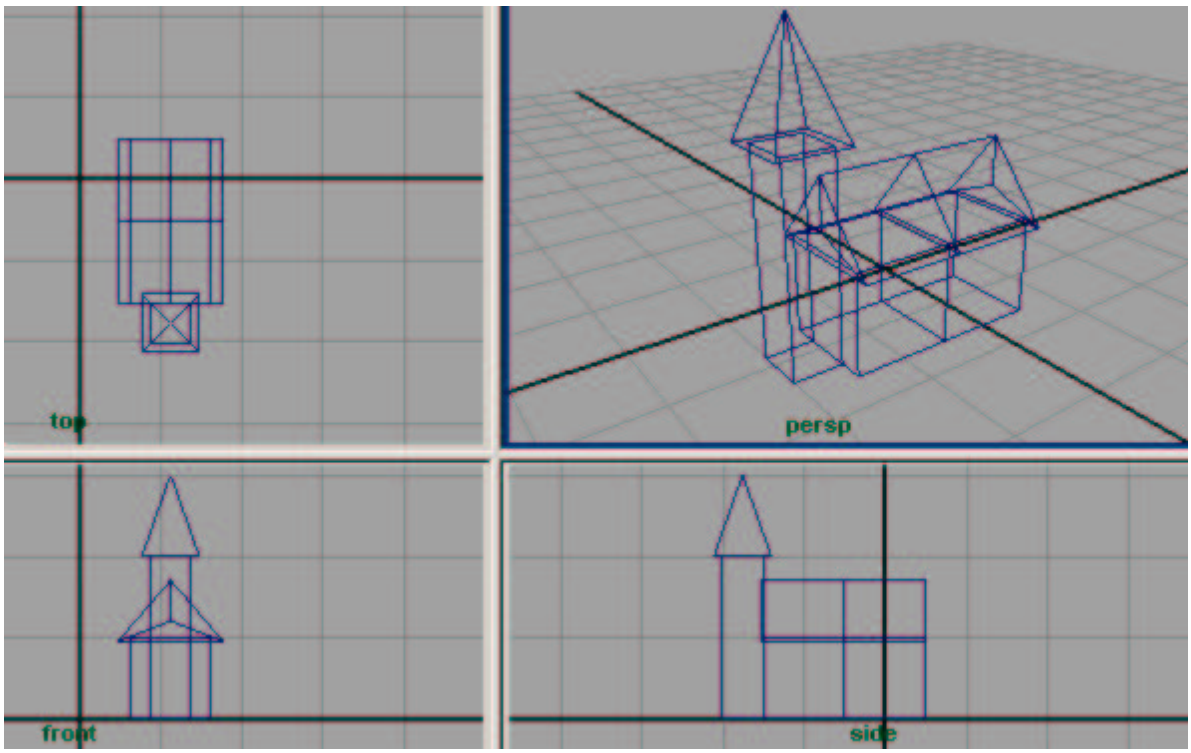

Figure 2.11: Many modeling software tools provide multiple views of a scene.

The projection matrix for the orthographic projection with a truncated viewing volume is defined by

$$P_{box} = \begin{bmatrix} 2/(R-L) & 0 & 0 & -(R+L)/(R-L) \\ 0 & 2/(T-B) & 0 & -(T+B)/(T-B) \\ 0 & 0 & -2/(F-N) & -(F+N)/(F-N) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The meaning of the parameters L, R, B, T, N, and F is the same as for P_{fr}. O_{box} is defined as long as $L \neq R$, $B \neq T$ and $N \neq F$.

Axonometric Projection

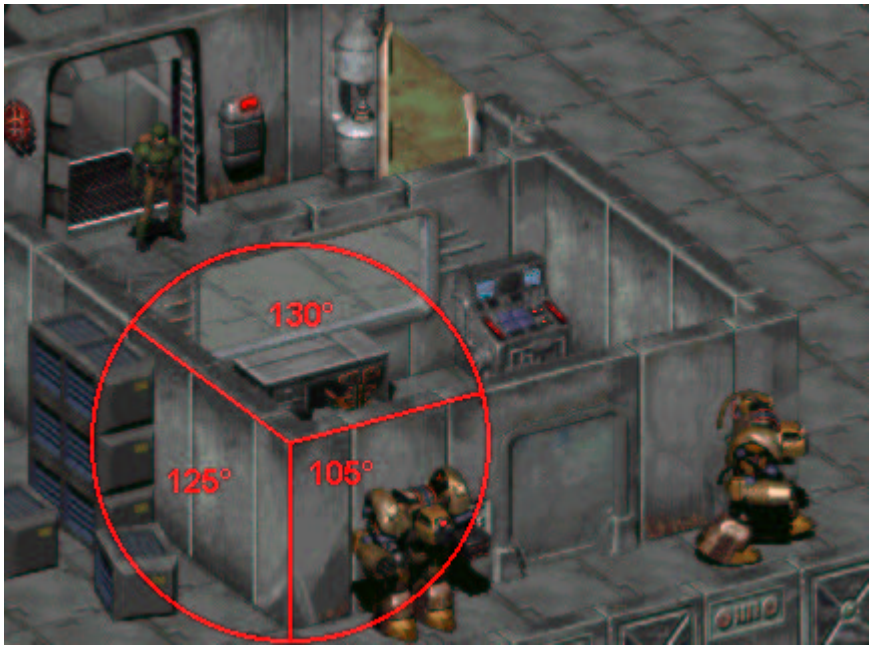

Figure 2.12: An example for a trimetric projection. Screenshot taken from [Int98], modified.

The axonometric projection is a special case of for the orthographic projection. The goal is to show as many surfaces of an object as possible. For a cube there would be three surfaces (using a parallel projection). To achieve this goal, the projection plane is rotated (the projectors remain orthogonal to the plane). Such an arranging can be done with a camera transformation, because the position of the projection plane is relative to the camera as previously mentioned.

Depending on the rotation of the projection plane the angles between the projected principal axes of the model vary resulting in specific scale ratios along these axes. These angles and ratios lead to the following classification: An axonometric projection is called isometric, if all three angles between the projected principal axis are equal (120°). In this case the same scale ratio applies to each axis. It is called dimetric, if two of the angles are equal. Then the scale ratio along two axes is the same. The projection is called

trimetric, if all angles and scale ratios are different. The isometric projection illustrates the 3D nature of an object and makes it unnecessary to use a multi projection while still allowing accurate measurements. However, it is more useful for rectangular shapes than for curved ones. Isometric projections are used for catalog illustrations and structural design. Axonometric projections are used in strategic and role playing computer games, where a lot of architecture can be found. In *Fallout 2* [Int98], for example, a trimetric projection is used (see Figure 2.12).

Oblique Projection

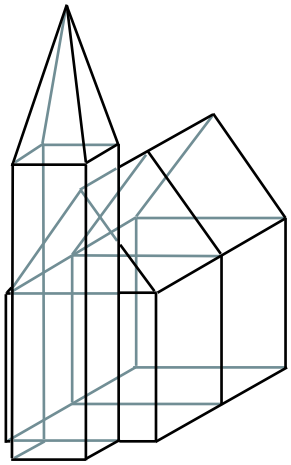

Figure 2.13: An example for a cabinet perspective. The length of lines perpendicular to the projection plane is reduced by a half in the projected image.

The oblique projection is a parallel projection with the projectors not perpendicular to the projection plane. It is defined by

$$P_{oblique} = \begin{bmatrix} s_x & 0 & \cos\beta/\tan\alpha * s_z & 0 \\ 0 & s_y & \sin\beta/\tan\alpha * s_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where α is the angle between the projectors and the projection plane, β is the angle defining the slope of the projection of lines that are perpendicular to the projection plane, and s_x, s_y, s_z are the scaling factors for the x, y, z components of a vertex. The value for α determines a factor s_o by which the lines that are perpendicular to the projection plane are scaled in the projected image. Typical values for β are 30°or 45°.

Two quite common types of the oblique projections are the *cavalier perspective* and the *cabinet perspective*. For the cavalier perspective $\alpha = 45.0°$ results in a scale factor $s_o = 1.0$. For the cabinet perspective $\alpha = 63.4°$ results in a scale factor $s_o = 0.5$. An example for a cabinet perspective is given in Figure 2.13. Beside the other properties of

parallel projections the knowledge about s_o allows accurate measurements in an oblique projected image. That is why these projections are used for technical illustrations.

The *stacking* projection, introduced in [MCMW01], is another type of the oblique projection. For this type the angle β is fixed to 90.0°. Points that are further back appear further up in the projection, depending on their distance to the camera and the angle α.

The viewing volume for oblique projections is a (infinite) parallelepiped.

Chapter 3

Related Work

In the following some work that has been done in regards to the creation of multi projection images, the simulation of eastern perspective, and the simulation of Chinese drawing styles will be introduced.

3.1 Multiple Projections

Figure 3.1: Replication of *Giorgio de Chirico's The Mystery and Melancholy of a Street.* Taken from [AZM00].

Agrawala et al. [AZM00] devised a system that allows different parts of a scene to be rendered with different cameras and then composed into a single image, called multi

projection. This process mimics art created by Cubist and Surrealist painters such as *Picasso* and *Dali*. Figure 3.1 shows a replication of *Giorgio de Chirico's The Mystery and Melancholy of a Street* done with their system. They attach every object in a scene to one of many cameras. Each camera renders the objects to which it is attached, then the images are layered together based on ordering provided by a "master" camera. This allows the master camera to determine the global ordering of objects, while the other cameras determine local orderings and perspectives.

The system by *Agrawala* et al. works quite similar to the application introduced in this thesis. Both systems create layers for each camera which are then composed into one image.

In [Sin02] *Singh* uses multiple projections to create a non-linear perspective projection. Multiple cameras are aimed at different parts of a scene. For each point of the scene a weight vector is calculated, indicating the influence of each camera on that point. Thereby, points that are close to the focal point or viewing direction of a camera are influenced mostly by this particular camera. A specific point is then projected using a projection which results from the average of all specified projections weighted by the weight vector for that point. Figure 3.2 contrasts the perspective projection with the non-linear perspective projection. The second one can be used to emphasize certain features of a model.

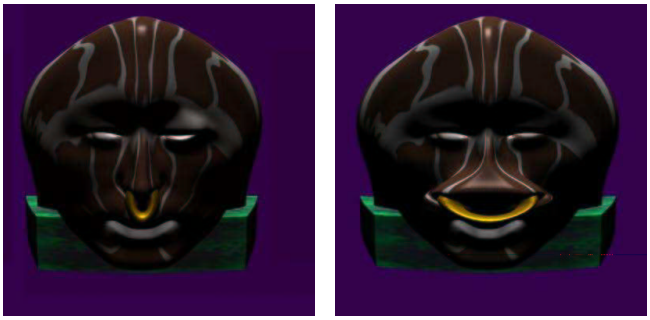

Figure 3.2: Single linear perspective projection (left), and non-linear perspective projection as the combination of three cameras: one on the left side, one on the right side, and one emphasizing the nose. Taken from [Sin02].

3.2 Eastern Perspective

Mason et al. introduce a system [MCMW01] that allows an artist to model a scene in the style of Japanese *seigaiha* with the help of AI methods and render it using different types of projections. During the modeling process 2D objects representing mountains and waves are placed parallel to the image plane in the 3D space. The *stacking* and *recession* projections introduced by the authors take the distance of an object to the projection plane into account and use this distance to modify the position of the projected object in

the image as shown in Figure 3.3 . This can be used to put objects that are further back from the projection plane further up in the image. Their approach produces nice results for 2D objects in a 3D environment. But it cannot be directly adopted to 3D objects in a 3D space. Using their projections for 3D objects would not stack the complete object but each single vertex. Thus the top surface would become (more) visible than desired.

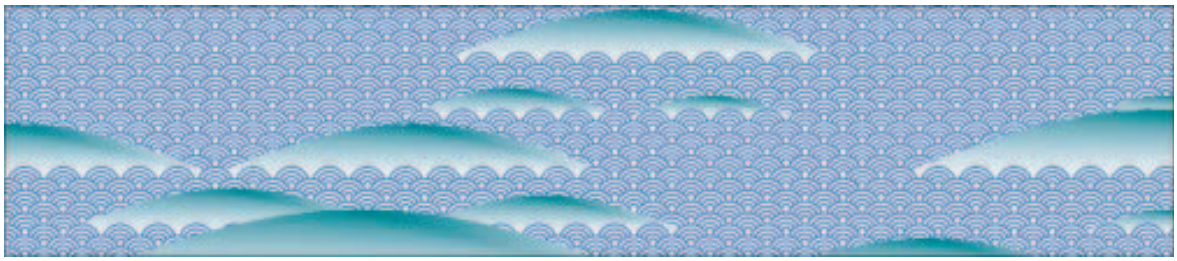

Figure 3.3: An example for a scene projected with the *stacking* projection. Taken from [MCMW01].

In [FCL03] *Farbiz* et al. describe an algorithm which converts photos into images that look like Asian art. In the first step the objects of the input image are segmented using image processing techniques. After this previously occluded object parts are filled using fuzzy averaging methods. The objects are then rendered in an Asian style using brush strokes. Finally the rendered objects are redistributed creating vertical gaps. Figure 3.4 illustrates this process. Similar to the work by *Mason* et al. [MCMW01] the idea of further back means further up is addressed. But this is only one of the many concepts for the composition of Chinese Landscape Paintings. On the other hand this work gives an impression of the combination of simulating composition techniques and drawing techniques. Combining both creates more pleasing and authentic results than simulating only one.

Figure 3.4: The steps for converting photos into drawings. Taken from [FCL03].

Figure 3.5: Mountains displayed using *hemp-fiber* strokes. Taken from [WS01].

3.3 Simulation of Chinese Drawing Styles

Hemp-fiber and *axe-cut* are two major types of texture strokes used for painting rocks in Chinese Landscape Painting. *Way* and *Shih* present methods for synthesizing these on the computer [WS01]. They simulate the motion of a brush as well as ink effects on paper. A user has to specify the contour of a mountain and then to apply the texture strokes. The strokes can be varied by a wide range of parameters describing the brush and ink properties. Figure 3.5 gives an example for mountains displayed using *hemp-fiber* strokes.

Trees are another important feature in Chinese Landscape Painting. The automatic generation of textures for 3D polygonal models of trees is the topic of [WLS02]. To create the outline of the tree the model's silhouette is computed depending on the position of the viewer. The silhouette is then used to create paths around the model which are drawn using user defined brush strokes. To create the texture for the bark different reference maps, such as depth map, normal map and curvature map are computed. This information is then used to determine the density for the placing of brush strokes. It can be varied by the user to simulate different drawing styles as well.

Combining methods for the simulation of drawing styles with multi projection images would allow to create more authentic images. A first step in this direction was already done in [FCL03].

Figure 3.6: Polygon model of tree textured with an automatic generated brush stroke texture. Taken from [WLS02].

Chapter 4

Requirements and Conception

In this chapter a conception for an application for the creation of multi projection images is derived. Starting from the basic requirements a rendering pipeline for multi projection images is introduced. Furthermore algorithms for modifying projections are explained. Finally a concept for the basic application layout is given.

4.1 Basic Requirements

The goal of this work is to implement a software that allows a user to create multi projection images from a given 3D scene. Different types of planar geometric projections shall be integrated and interaction methods for working and adjusting this projections shall be provided. The inspiration for this work are Chinese Landscape Paintings with their multiple projections and viewpoints in a single image.

Out of it the following aspects have to be addressed:

- A modified rendering pipeline has to be designed and implemented since the traditional one only supports the use of a single projection at a particular time. Beside the commonly provided perspective and orthogonal projections, the oblique projection needs to be provided.

- Giving the user full control over parameters of the projections is a major demand. An appropriate user interface needs to be designed and implemented.

- Interactivity is a requirement that goes along with the requirement of control. This means the user should get immediate feedback on actions that have been done for modifying the scene. The control mechanisms should be simple, intuitive and meaningful.

- Finally the user should be able to save the resulting scene composition. Beside the ability to create screenshots of the rendered images, the settings for the creation of

a particular image should be storable. This allows to reproduce a particular effect as well as to use these settings as input for the program.

4.2 A Rendering Pipeline for Multi Projection Images

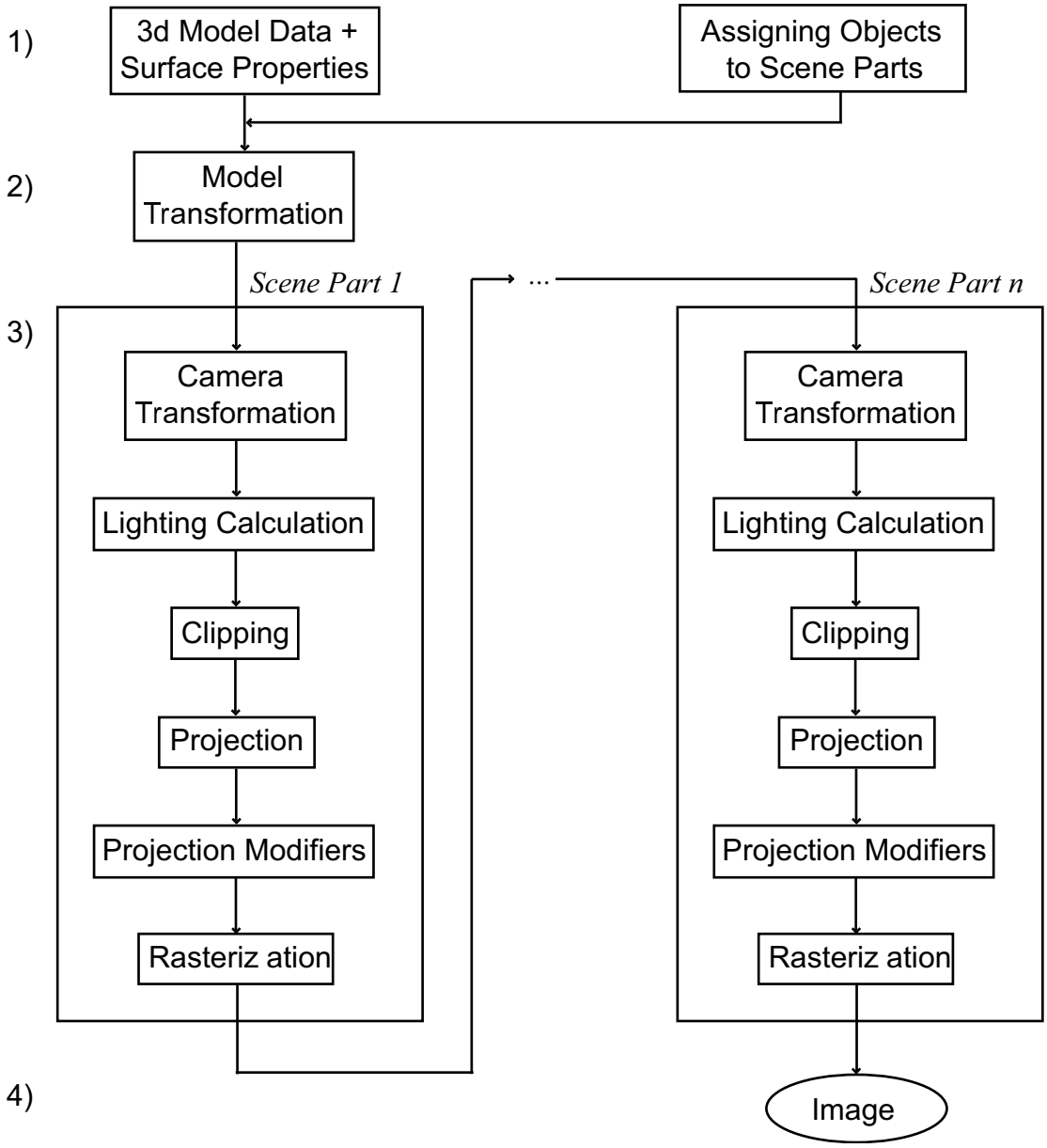

Figure 4.1: A rendering pipeline for multi projection images.

From the analysis of Chinese Landscape Paintings in Section 2.1 it becomes apparent

that the areas that are projected in different ways can be treated as different layers. The attempt to simulate this leads to a new rendering pipeline, illustrated in Figure 4.1:

1. The input for the pipeline is the usual model data: geometry information as well as surface data such as material and texture. Furthermore each object is assigned to a specific scene part. A scene part is a subset of the model. In the following process each scene part can be different from the others. If there is only one scene part, the rendering process is basically similar to the classic rendering pipeline introduced in Section 2.2.1.

2. Comparable to the standard rendering process model transformations such as scalings can be applied.

3. After this all scene parts will be processed sequentially:

   - The camera's position and orientation for the actual scene part is applied.
   - Then the lighting calculations using the object's surface properties are performed.
   - Optionally all non visible objects can be removed to reduce rendering costs in the following steps. This could be done by testing the objects against the viewing volume.
   - The according projection for the scene part is applied.
   - After the actual projection modifiers can be applied to change the projection's position in the final image. Such modifiers are discussed in section 4.4.1 and in section 4.4.2.
   - The result is drawn into the framebuffer.

4. The result of this process is a multi projection image.

4.3 Image Composition

In the previously described rendering process an image is created for each part of the scene. These images are then placed over each other, creating the final image. As the layers are independent from each other the idea of the *painters algorithm* [FvDFH90] can be used to resolve the visibility among them. The *painters algorithm* is a simple HSR technique to resolve visibility among objects of a scene: the objects are sorted by depth and are then drawn from back to front. Occluded parts of objects that are further away are overdrawn by closer objects.

For the scene parts that means, those that should appear further away are painted first and closer ones are drawn in front of them. To resolve the visibility among the objects of a scene part *z-buffering* seems to be appropriate.

Figure 4.2 gives an example for the creation of a multi projection image: the scene is displayed using two cameras at different positions. Each camera only projects the objects that are assigned to its scene part. The first scene part consists of a pyramid that is shown from the front. The two cubes in front of it are projected by the other camera which is viewing down on them from a higher position. The two resulting images are drawn over each other starting with scene part 1, the background object.

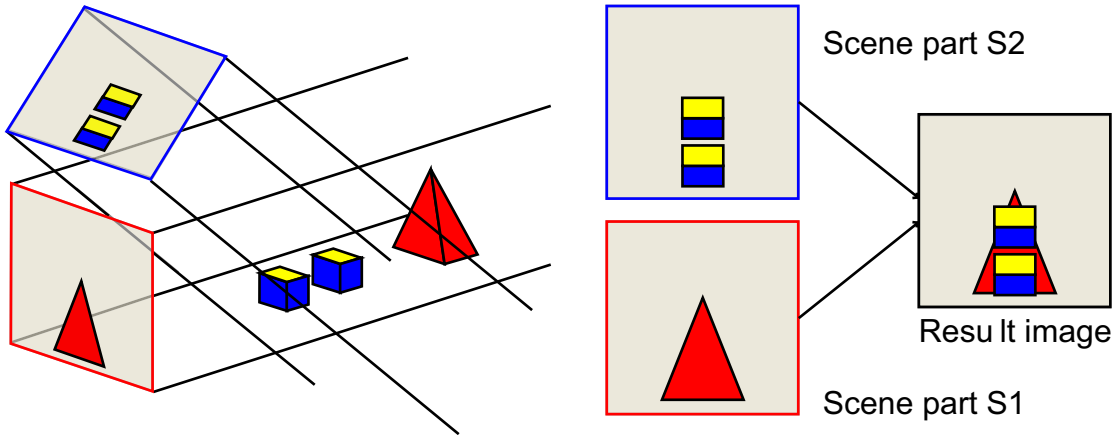

Figure 4.2: Image composed of two scene parts using different viewpoints.

But to be able to create a multi projection image that way it is first necessary to divide a scene into scene parts, to assign objects to them, and to specify camera settings and projections for them. Therefore it is necessary to be able to:

- Create new scene parts.
- Assign objects to a scene part. It might be possible to choose the objects that should belong to a scene from a list naming all the scene's objects. But it seems to be more intuitive to select the object from a graphical representation of the scene. Selecting single objects can be done by picking. Groups of objects can be selected using rubberbanding. Objects that are assigned to one scene part should be removed from all others.
- Specify camera transformations and a projection for a scene part.
- Creating an order among the scene parts. As already mentioned, the order in which the scene parts are rendered has a huge influence on the resulting image.
- Remove objects from a scene part. It might be desired to completely remove objects from the image. As for assigning objects the methods of picking or rubberbanding can be used.
- Delete scene parts. Unused scene parts should be deleted to keep the list concise.

4.4 Projection Modifiers

In the following two techniques to modify the x any y position of a scene part in the resulting image are presented. They have no influence on the ordering of the scene parts.

4.4.1 Post Projection Shifts

An essential aspect is to allow the user to modify the position of a scene part's projection in the image. For some parallel projections this could be done by changing the camera position. However, for perspective projections such a change would also change the shape of the projected objects. Figure 4.3 illustrates how the position of the camera can influence the shape of the projected object.

The image illustrates the setting for projecting a cube using a perspective projection. On the left side the cube is located close to the upper side of the viewing volume. Thus its projection is located in the upper part of the projection plane, showing three of its faces. If the projection should be brought in another position on the projection plane using a camera transformation, this will result in a change of the shape of the projection. The right side of Figure 4.3 shows that after translating the camera in a way that the object is in the center of the viewing volume, also its projection is in the center of the projection plane. But only one surface remains visible.

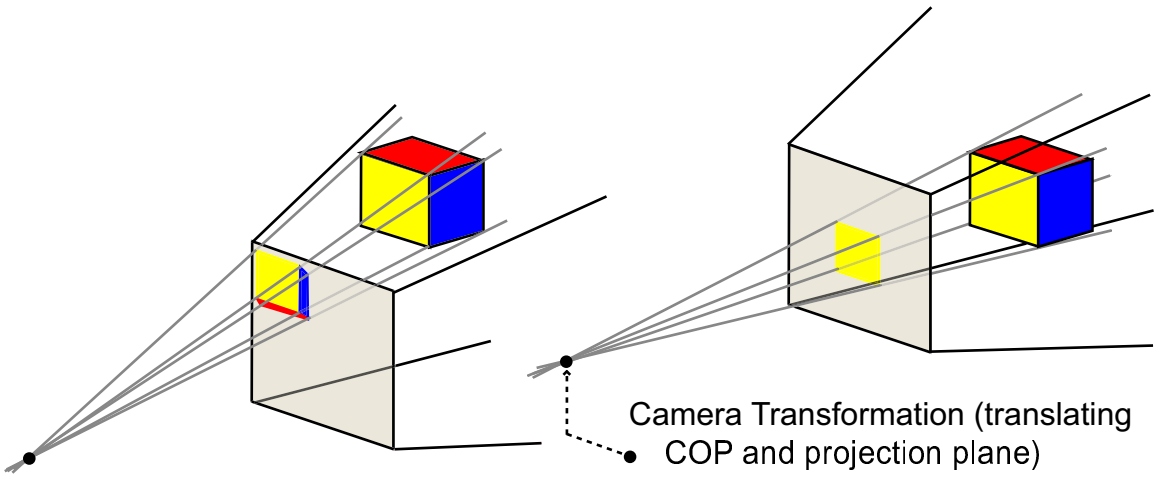

Figure 4.3: Using camera transformations to adjust the position of an object in the image can result in a change of the shape of the projected object.

To circumvent this problem translations can be applied after the projection moving the projected object on the projection plane. These post projection shifts can be applied in x- and y-direction in the projection plane, as illustrated in Figure 4.4: rather than using a camera transformation, a translation is applied after the projection to move the projection of the object on the projection plane. Thus the shape of the projected object remains unchanged (right).

The modified projectionP' of a point P computes as $P' = T_{xy} \cdot Pro \cdot CT \cdot MT \cdot P$. With MT as the model transformation, CT as the camera transformation, Pro as the projection, and T_{xy} as the post projection shift given as translation in x and y.

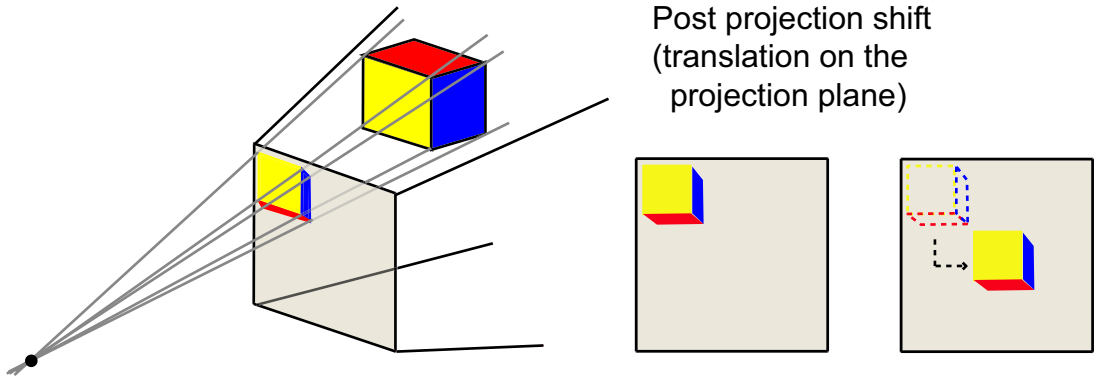

Figure 4.4: Using a post projection shift to adjust the position of an object in the image preserves the shape of the projected object.

4.4.2 Dependencies between Scene Parts

An attempt for automatically positioning scene parts in the image is to create dependencies between them. Assuming a (imaginary) ground plane on which the objects are located, this plane would usually be disrupted for scene parts using different viewpoints or projections. Unwanted gaps or overlappings would occur between scene parts in the composed image as illustrated in Figure 4.5(a).

An intuitive approach to circumvent this is to virtually divide the plane while making sure that the two parts remain connected. Discontinuities resulting from different types of projections or from different viewpoints caused by camera rotations around the x-axis can be solved as illustrated in Figure 4.5: assuming two scene parts S_1 and S_2 there S_1 is a background scene part that should be aligned to the foreground scene part S_2. At first the furthest point P of S_2 is determined. Then P is projected using the projection of S_1 resulting in a point A. Projecting P using the projection of S_2 results in point B. If the projected images of S_1 and S_2 are put together without any modifications the points A and B are at different positions (see Figure 4.5(a)). Applying the y-distance d between the two points as a translation to the projection of S_1 removes this discontinuity in the result image (see Figure 4.5(b)).

As previous scene parts could have effected S_2 these affects need be taken into account also. Thus a cascading influence of closer scene parts to further ones is created.

The idea only works properly if the involved cameras are rotated just around the x-axis. Further rotations would result in discontinuities that cannot be solved clearly: assuming two scene parts projected in the same way, but with one camera rotated around its z-axis. Thereby the planes cross each other in the image leaving gaps between each

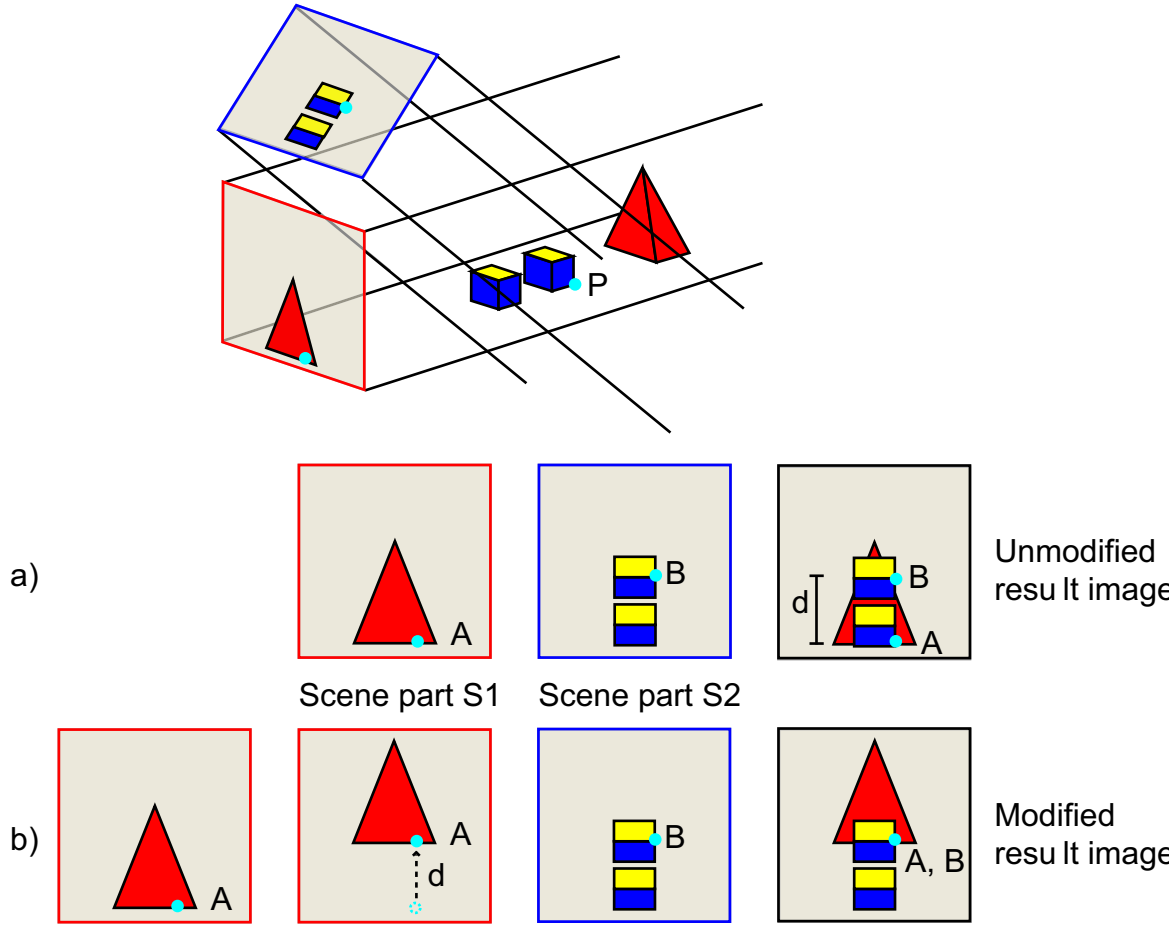

Figure 4.5: Image composed of two scene parts using different viewpoints. Composed without modification (a) and with dependency between the scene parts (b).

other as shown in Figure 4.6. Translating one projected part in y-direction would only make one gap smaller and the other one bigger.

The furthest point P in S_2 needs only to be determined, if an object is added or removed to the scene part. The dependency between two scene parts only needs to be updated, if the camera transformation or projection of one scene part is changed.

To introduce gaps or overlappings between scene parts for artistic reasons, it should be possible to combine post projection shifts with the cascading effect as already incorporated into the algorithm. Using the same ideas, a dependency of the foreground scene part to the lower image border can be created, to align that scene part at the bottom of the image.

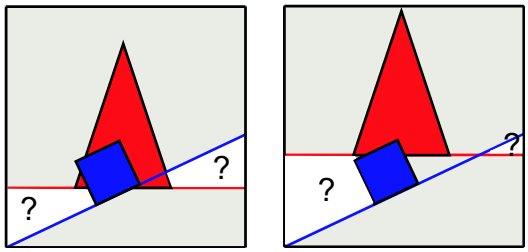

Figure 4.6: Rotating one camera around the z-axis creates gaps between the (imaginary) ground planes.

4.5 Picking in Multi Projection Images

The most intuitive way to create and apply a post projection shift for a scene part would be to select an object of the scene part in the result image and move it to the desired position. The shifting could then be calculated from the difference between the initial mouse position and the final mouse position transformed from the screen space into the object space. To be able to do this picking in a multi projection image is necessary. The following algorithm shows, how this can be done:

Input: Mouse position
Output: O, list of objects at this mouse position
H, array of number of hits for each scene part

begin
  $O := \emptyset$
  $H := \emptyset$
  for $i := \#(scene\ parts)$ **to** 1 **do**
    $m := 0$
    foreach object of scene part i
      if object is hit by mouse
        $O := O + \{\{object\ id,\ object\ information\}\}$
        // object information such as the depth of the hit occurrence
        $m := m + 1$
      endif
    endforeach
    $H := H + \{m\}$
  endfor
end

As illustrated in Figure 4.7 it is easy to get a fast correlation between an object and a scene part. By using the information from the list H it is also easy to find the objects that were hit for a specific scene part in list O. In terms of scene parts the list O is sorted by depth starting with the closest scene part. If more than one hit occurred for a scene

part the depth information about the objects can be used to create an order among the objects of that scene part.

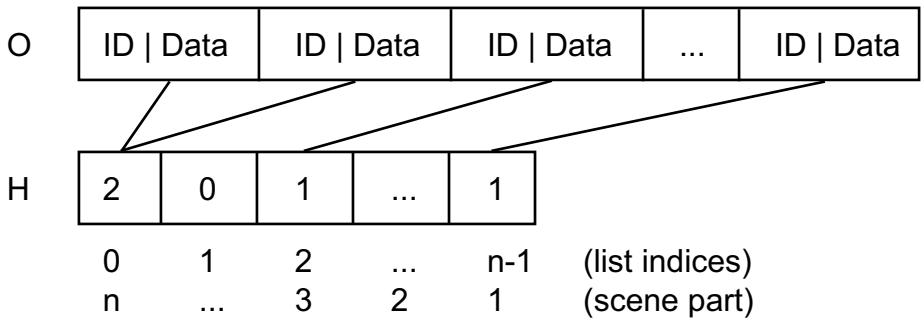

Figure 4.7: The correlation of the list O containing the objects and the array H with the number of hits that occurred for the scene parts.

In case of applying a post projection shift to a scene part by selecting an object and moving it, the algorithm can be stopped as soon as the first object hit was found. The actual value of i names the accordant scene part. As closer scene parts are drawn in front of further scene parts the algorithm starts to test the closer scene parts first. Thus the first hit that occurs then belongs to an actual visible object.

4.6 Object Stacking

As already mentioned the concept of putting objects that are further back further up in the image is quite common to express depth. The image shown in Figure 4.8 is an example for objects that are put higher the further away they are. All those hills, mountains, and isles are seen from the front, their top surface is not shown. In [MCMW01] Mason et al. introduce the *stacking* and *recession* projections to mimic such effects. As these projections create nice results for 2D objects that are placed parallel to the projection plane they are not appropriate to create an effect such as in Figure 4.8 with 3D objects. For 3D objects the *stacking* and *recession* projections would also show the top surfaces.

To avoid this the stacking effect needs to be applied to objects only and not to each vertex as the projections mentioned above do. The following algorithm creates a simple object stacking effect for a scene part:

Input: Scene part S
i, interpolation coefficient
s, stacking coefficient
Output:

begin
foreach object of S

Figure 4.8: *Wen Chia's Landscape in the Spirit of Verses by Tu Fu* as an example for object stacking. Taken from [Cah77].

```
    compute closest point A and furthest point B of the object
    project object
    translate projected object in y-direction by ((B_z − A_z) ∗ i + A_z) ∗ s
    draw transformed object
  endforeach
end
```

It seems to be most intuitive to use an interpolation coefficient $i = 0$ or in other words to use the front of an object to determine the amount of stacking. Depending on the type of model the algorithm can produce undesired results: e.g. if a smaller object is placed in the middle on top of a larger object touching that, it will appear flying over the bigger object after the projections. This results if each object is treated independently. To solve the problem either hierarchical models might be useful or the stacking might be applied on scene parts and not on objects of one scene part. Both solutions could require higher efforts either in the modeling process or in the image composition process creating the scene parts. A further problem is to adopt the previously discussed concept of picking and scene part dependencies to the object stacking. During the implementation only the simplest version of object stacking was realized to demonstrate the basic idea. Neither

picking nor an integration into the dependency concept were implemented, due to the poorly conceived state.

4.7 A Concept for a User Interface

The previously discussed aspects yield that the application to be implemented will have three main groups of components. First of all an area to present the render results is needed. In this component the user should be able to rearrange the projected scene parts by interactively applying post projection shifts. Secondly manual controls should be provided which allow the management of a list of scene parts and the settings of an individual scene part. The camera and projection parameters as well as the settings for the introduced algorithms belong to this. Finally a separate view of the scene should be provided, in which the user interactively can assign objects to scene parts and control the camera and projection settings.

As shown in Figure 4.9 three possibilities for placing these groups of components in an application exist:

- In a *Single Window Application* each component would be embedded in the application's main window. The advantage of this is to have all components visible all the time. But on the other hand it limits the size for the visual feedback as all components have to share the same screen. Making the components resizable would reduce this problem, but it would not eliminate it.

- In a *Multi Document Interface Application* would be again only one main application window. Inside this main window several subwindows can be placed. These windows can be put at any position inside the main window. They can be resized as well as be displayed in full size. This allows a user to have all desired components visible at a desired size. If necessary or desired one component can be displayed in full size. These feature make a MDI application more flexible than a single window application.

- In a *Multi Window Application* each group of components is placed in a separate window. Comparable to the MDI application, each window is fully resizeable. For workstations with more than one monitor it adds some additional flexibility as the less used windows can be placed on an auxiliary monitor. In contrast to using dialogs for the controls, the usage of independent windows is more flexible (e.g. a window creates an own taskbar entry which allows easy accessing).

As each of the component groups will consume a relevant amount of space a multi window application is most appropriate. It provides the highest amount of freedom for arranging the windows on one or more screens.

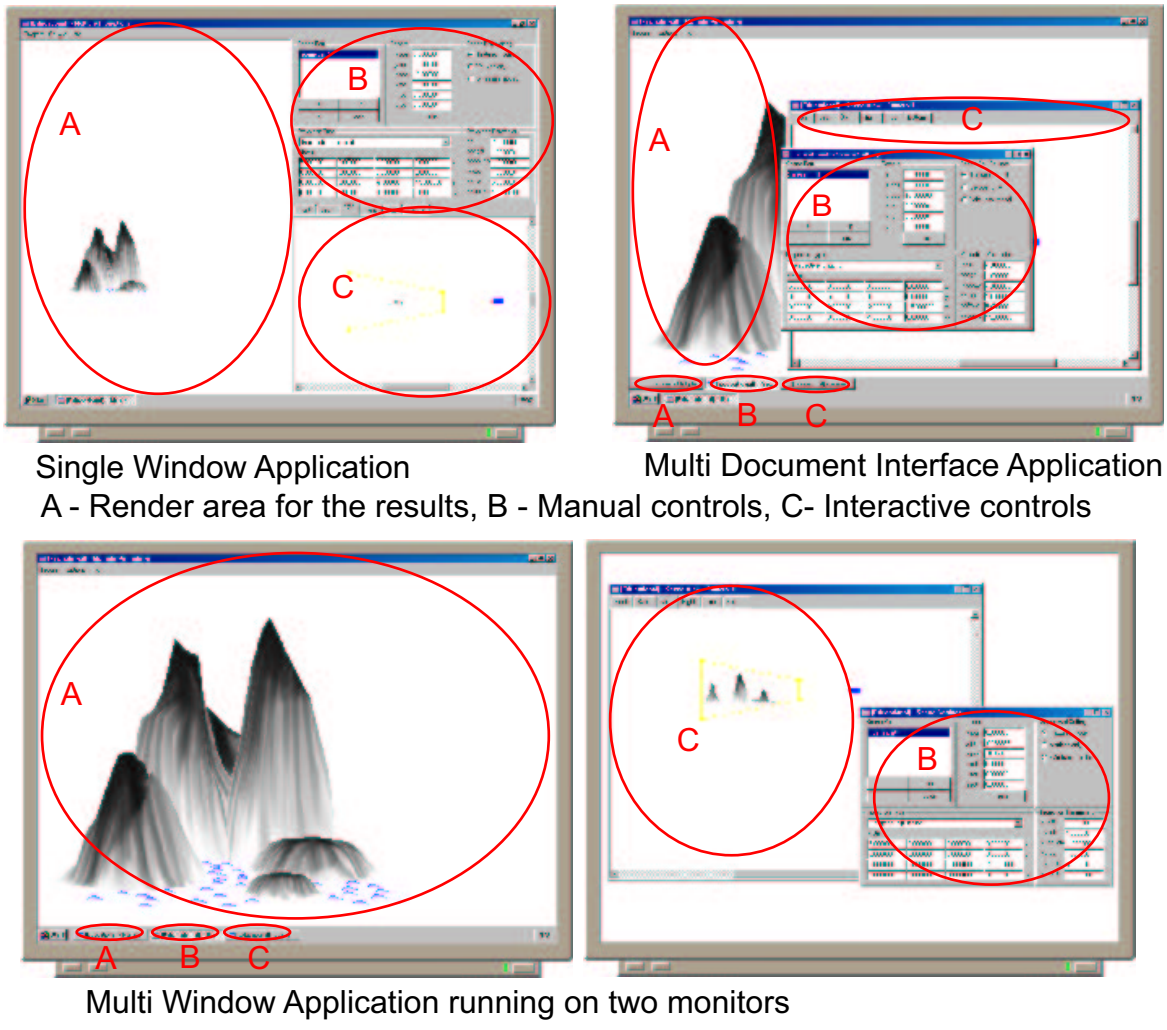

Figure 4.9: Three possibilities for the component layout.

Chapter 5

Implementation

Based on the conception given in the previous chapter an application called MPI-Builder was implemented. In the first section of this chapter the resources that were used for the implementation are introduced. After that some auxiliary classes the application is based on are explained. The application itself is discussed by its three main components: the Main Window, the Scene Part Settings Window, the Scene View Window. At the end of the chapter different result pictures created with the MPI-Builder are presented.

5.1 Resources

The general coding of the program was done in the C++ programming language. The graphical rendering was implemented in OpenGL, and the user interface was realized using Trolltech's Qt API. In the following these three tools are introduced shortly:

- *C++* was chosen as programming language for the project. This object-oriented language is the standard for creating high-performance applications. C and Assembly source code can easily be integrated into C++ code allowing even direct access to hardware. Many APIs are available extending the basic functionality provided by the language. Besides some restrictions, C++ source code can be compiled on many platforms. For a project of this size these restrictions barely apply.

- OpenGL is the software interface to graphics hardware for developing portable, interactive 2D and 3D graphics applications. It is designed as a state machine consisting of about 150 distinct commands that are used to specify the objects and operations. The programmer can use these routines to manipulate the state machine to generate the desired graphics output. Furthermore OpenGL is designed as a hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL [WNDS99].

Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics API.

- For the implementation of the user interface Trolltech's [Tro04] application framework Qt was chosen. It is written in C++ and completely object-oriented and intuitive. Qt provides a huge set of widgets. These are components (e.g. windows or buttons) for the creation of a user interface. Existing widgets can be extended easily by subclassing. New implemented widgets can be integrated as well. The framework also supports the usage of OpenGL by providing widgets that allow the displaying of OpenGL context. Beside these properties Qt also provides additional functionality, particularly the functions for image handling turned out to be useful. Finally the same Qt source code can be compiled on Windows, Linux/Unix, Mac OS X, and embedded Linux creating native code on each platform.

  The communication between Qt objects works different then in other GUI tool kits. Those use callbacks for each action that can be triggered. These callbacks are pointers to functions. In Qt the communication is done by a signal/slot mechanism: a signal is emitted by an object when its state changes. A signal will be received by all slots its connected to. A slot is just a normal member function with the capability to receive signals. Both, signals and slots, can take any number of arguments of any type.

The coaction of the three components, the capability to create high-performant software with them, and the possibility to write source code which compiles on different platforms lead to the decision to use them. During the implementation Windows as well as Linux were used, resulting in source code that can be compiled on both platforms.

5.2 Base Classes

5.2.1 3D Input

For loading 3D model data into the program, Alias Wavefront's object file format OBJ with the associated material file format MTL was chosen. On the one hand source code for loading those files is available, on the other hand OBJ files may be created and exproted by using Maya. Finally both formats contain ASCII text, that would allow to easily modify the files if this would become necessary. The OBJ/MTL loader used in the MPI-Builder is based on the source code of *Nate Robin's* GLM library [Rob97].

An OBJ model consists of several groups. One material is associated with each group. Each group is composed of triangles. The loader assigns a unique object id to each group to allow the selection of objects. Each group and material is stored in a separate OpenGL display list, which allows efficient rendering and displaying of the objects.

5.2.2 Scene Part

The class MPIBScenePart is the most important data structure for the application. An array of boolean values is used to indicate, which objects of the model belong to it. The class stores the camera's position and orientation- The projection is managed in a separated object, which can be accessed using a pointer stored in the class MPIBScenePart. These information is necessary for the basic rendering process. The parameters for the additional algorithms is stored as well in this class.

5.2.3 Projections

In the application a projection is represented by two classes. The first one contains the projection matrix, a name to identify its type, and a set of parameters. The class is responsible for calculating its projection matrix based on its parameters. Furthermore it provides functions for drawing an appropriate viewing volume (that is used in the Scene View Window) and to react on interactions on the viewing volume. The second class provides a widget that allows the user to access the projection's parameters.

The following projections were implemented:

- MPIBProjection is the base class for all other projections, providing the interface for accessing a projection from the other classes of the application. It does not have any parameters beside the projection matrix and a name. It can be selected to be the projection of a scene part. The values for its projection matrix can then directly be edited in the Scene Part Settings Window.

- MPIBPerspective and MPIBPerspectiveClipping implement the perspective projection given by the matrices P_{persp} and P_{fr}. The second one is similar to the matrix that would be created with OpenGL's function *glFrustum*.

- MPIBOrthographic and MPIBOrthographicClipping implement the orthogonal projection given by the matrices P_{ortho} and P_{box} i. The second one is similar to the matrix that would be created with OpenGL's function *glOrtho*.

- MPIBOblique implements the oblique projection given by the matrix $P_{oblique}$.

- Although the stacking projection is an oblique projection and the recession projection is a perspective projection they are implemented in the classes MPIBStacking and MPIBRecession. Both use the concept of a stacking coefficient rather than angles to create the stacking effect as described in [MCMW01].

An axonometric projection was not implemented as the rotation of the projection plane can be achieved easily by rotating the camera.

5.2.4 QExtNumberEdit

Qt provides the widget QLineEdit, which displays a single line of text which can be edited by a user. To achieve some further desirable options the class QExtNumberEdit which inherits from QLineEdit was implemented. This widget has the look of its base class, but is designed for the input of floating point numbers only. To avoid the input of illegal datatypes the class uses a validator which declines any wrong input such as characters that are typed in by mistake. Furthermore the widget allows a user to quickly modify the actual value using hotkeys. The arrow-keys *Up* and *Down* can be used to in- or decrement the value by a small amount, the keys *PageUp* and *PageDown* add or remove a higher amount. These values can be set by the programmer according to the application's needs. Finally a lower and upper boundary can be set which the value can't pass. Two types of behavior can be specified to react on the crossing of such a boundary. Either the value can be reset to a specific value depending on the boundary, or a new value can be computed, adding the difference of the value and the crossed boundary to the opposite boundary until there is no more violation of a boundary. The second possibility is especially useful for the input of angles.

5.3 Main Window

The application's Main Window contains a MPIBGLWidget which is derived from Qt's QGLWidget. The MPIBGLWidget implements the rendering pipeline and algorithms introduced in Chapter 4. The class stores information about the scene in a list of scene parts. The content of that list is the input into the rendering process. The widget is also responsible for displaying the created graphics context. By using the picking algorithm for multi projection images it is possible to interactively apply post projection shifts.

Furthermore the Main Window provides a menubar, that allows a user to access the functions described below. The most important of these functions can also be accessed by hotkeys:

- The *Load Model* option (also accessible using *F1*) opens a file dialog in which the user can specify an OBJ model to be loaded. The new loaded model will be scaled, so that it fits into the unit cube, and the complete model is visible in the beginning. Loading a new model resets the list of scene parts to contain only one scene part with default settings, containing all objects of the scene.

- *Load MPI Scene* (also accessible using *F2*) opens a file dialog which allows a user to load a *.bss* file. A *.bss* file references an OBJ file and specifies scene part settings. In the preview window of the file dialog the related screenshot to the *.bss* file can be displayed, to help the user browsing through the files (see Figure 5.2). The actual parsing and the creation of the datastructures is done by the class MPIBGLWidget.

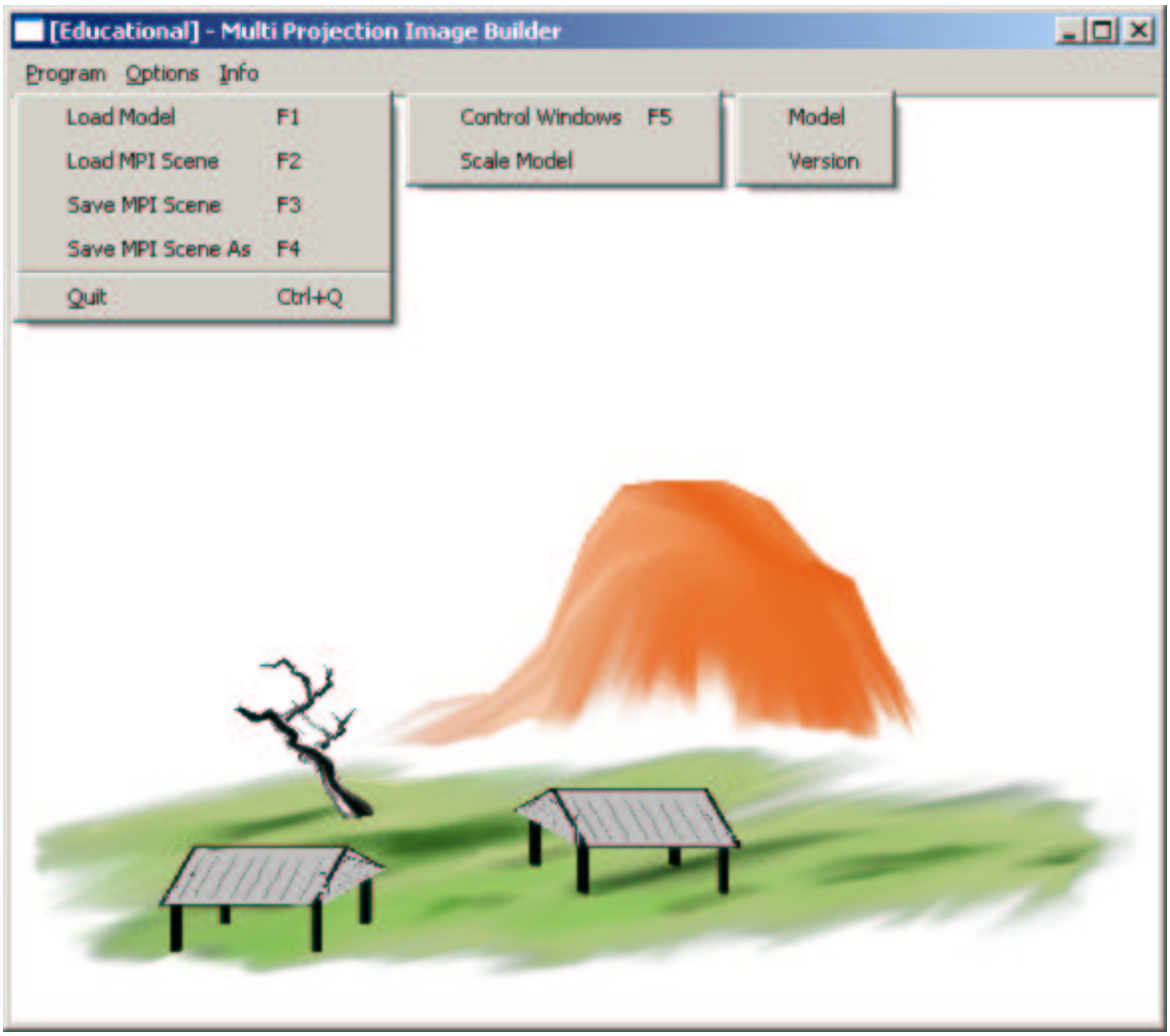

Figure 5.1: The Main Window contains the renderarea and provides a menubar.

- The option *Save MPI Scene* (also accessible using *F3*) reads the framebuffer for the MPIBGLWIDGET and saves it as bitmap. Additionally the scene part settings that were used to create this particular image are saved as a *.bss* file.

  Both files are stored in a folder which was created at the start of the program, containing the date and time of the creation in its name. The files themselves are named like "snapshot_001.*" with increasing numbers.

- Using *Save MPI Scene As* (also accessible using *F4*) opens a file dialog in which the user is able to specify a folder and file name, for the screenshot and *.bss* file.

- The option *Control Windows* (also accessible using *F5*) opens the Scene Part Settings Window and the Scene View Window which are explained in the next sections.

- *Scale Model* opens a dialog which allows the user to scale the model to the desired size.
- Information about the model (such as the number of objects, the number of vertices, etc.) and the program can be displayed.
- Finally the program can be closed using the *Quit* function, or the close button of the window.

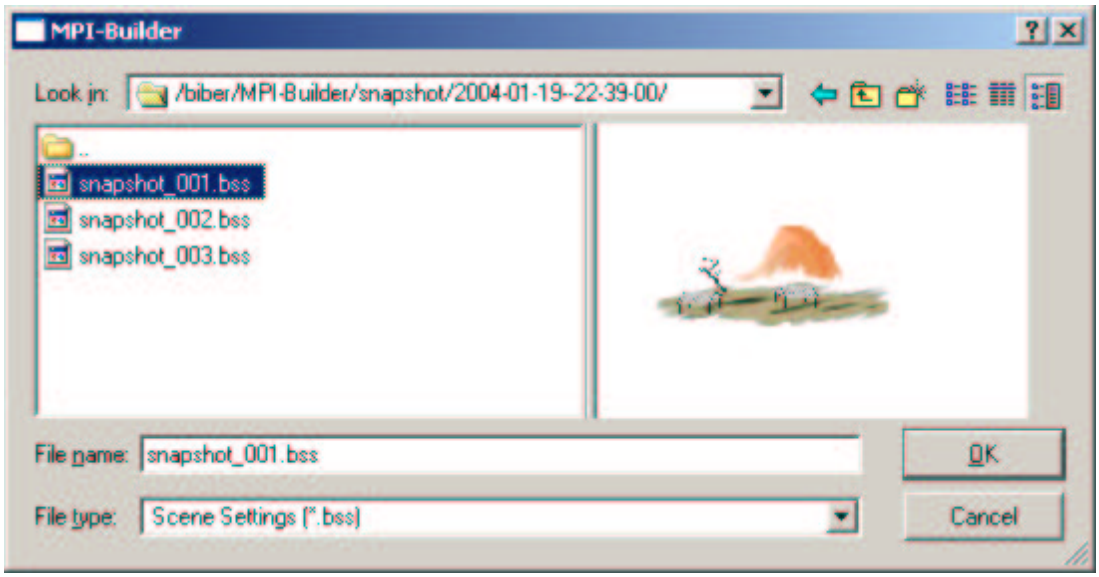

Figure 5.2: File dialog for loading *.bss* files. A preview generated from the according bitmap for the selected file is shown.

5.4 Scene Part Settings Window

Figure 5.3 shows a screenshot of the Scene Settings Window. Its components are described below. As widget for the input of floating point values the class QExtNumberEdit was used.

- In the Scene Part section in the upper left corner a list displays the names of all scene parts that were created. By selecting an entry in the list, the according scene part becomes active. Its settings are displayed in the other widgets of the window and can be changed there. The currently active scene part is always highlighted in the list.

  A new scene part can be created by pressing the "+" button that is located under the list. The new scene part will be put right behind the currently active scene

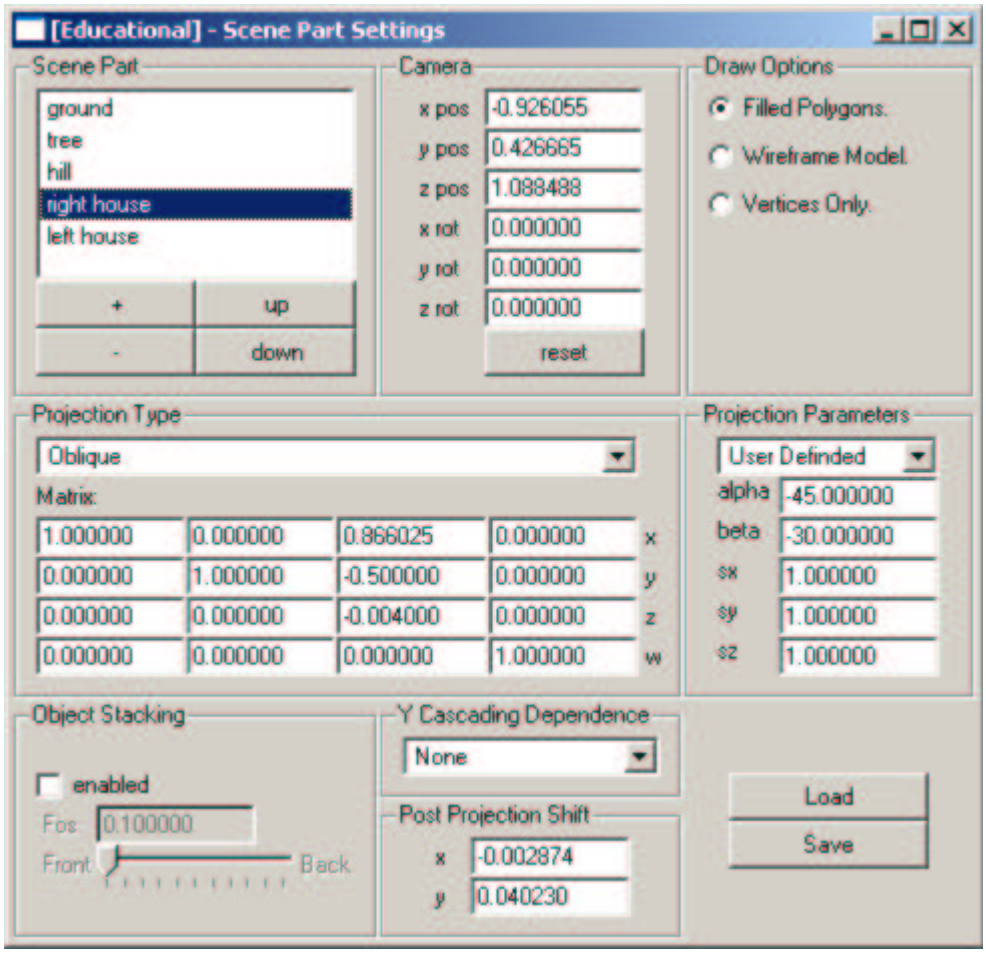

Figure 5.3: Scene Part Settings Window.

part in the list. After creating it, the new scene part becomes the active scene part. By pressing the "-" button the currently active scene part is removed from the list. The buttons "up" and "down" can be used to move the active scene part up or down by one position in the list.

Then a new scene part is created it gets a default name like "scene part 2". By clicking on the entry with the right mouse button a dialog is shown where the name can be changed.

- The Camera section right beside the Scene Part section provides three input fields for the camera coordinates and three input fields for the camera's rotation. The class QExtNumberEdit was used for that. The user can change the values and apply all of them by pressing the *Enter* key, while one of the fields has the keyboard focus.

  The "reset" button sets the camera's parameters to their default values and applies them.

- In the Drawing Options, located in the upper right corner of the window, the user can choose between three render styles for the scene part: If *Filled Model* (the default setting) is selected the polygons are drawn filled using the specified material information. If *Wireframe Model* is chosen, only the outline of the polygons are displayed. The third option draws only the vertices of the objects.

  These options are presented mainly to give the user a simple way to distinguish between different scene parts in the result image.

- From the combobox in the Projection Type section, in the middle of the window, the user can select the type of projection that should be used. The combobox displays the name of the chosen projection.

  Depending on the selected projection a widget is displayed containing input fields for the projection's parameters. As for the camera settings the parameters can be changed and applied by pressing the *Enter* key while one parameter field has the key board input.

  The projection matrix that results from these parameters is displayed in a field that is located under the combobox for choosing the projection's type. It is possible to directly change and the apply the values in these matrix fields. Note that this will not change the parameters in the projection widget.

- Object Stacking can be enabled using a check box. The coefficient determining the amount of stacking can be set in a field. The coefficient for interpolating between the front and the back point of an object can be set using a slider.

- A cascading dependency to another scene part can be created, by selecting this scene part in the combobox.

- Finally it is possible to access the functions for saving and loading the settings for a scene composition from this window, using the buttons in the lower right corner.

Any applied change on the settings results in an immediate rerendering of the scene that is shown in the Main Window. The view ofthe scene in the Scene View Window is updated as well.

5.5 Scene View Window

The Scene View Window provides a views of the scene, and shows the current scene part's camera and viewing volume at their relative position and orientation to the model. The user can select from six different views: front-, back-, left side-, right side-, top-, or bottom-view using a tabbar.

Objects belonging to the currently selected scene part are displayed in their original color. All other objects are displayed in blue marking them as deselected. A single

non-active object can be assigned to the current scene part by clicking on it with any mouse button. The same way an active object can be removed, without assigning it to any other scene part. Entire groups of objects can be assigned to the current scene part using rubberbanding: the user clicks on a point and moves the mouse to another point on the screen while the mouse button is pressed. A green transparent rectangle shows the selected area. When the mouse button is released every object inside the selected area is added to the scene part. In the same way entire groups of objects can be removed from the scene part. Therefor it is necessary to press the *Ctrl* key while interacting with the mouse. After selecting or deselecting objects their color is updated to match their status.

The position and orientation of the camera can be changed interactively as well. To translate the camera, the user needs to select it using the left mouse button. Then the camera can be moved until the mouse button is released. Using the right mouse button allows the user to rotate the camera.

The viewing volume of a projection can be adjusted using the manipulators on it. These little spheres can be selected with the mouse and translated, while any mouse button is pressed. The viewing volume (and the projections parameters) change according to the applied translations. The calculation for such changes are done in the projection class. It gets the object ID and the amount of translation, and calculates the change of the according parameters from these.

Finally the render area is embedded in a scrollable context and a zoom function for the scene is provided. This allows the handling of larger scenes.

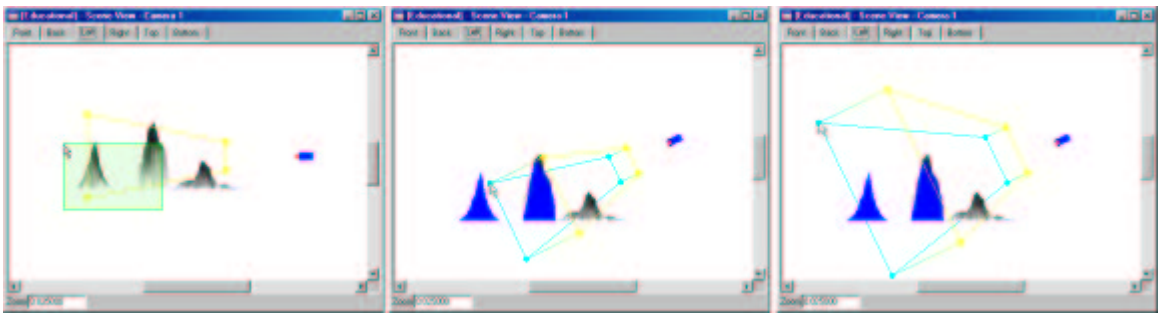

Figure 5.4: Scene View Window. Deselecting objects using rubberbanding (left). Interacting with the viewing volume (mid and right).

The screenshots in Figure 5.4 show views of a scene from the left side. The first screenshot shows the camera in the default position. All objects belong to the current scene part. Deselecting a group of objects is also illustrated: each object under the green area will be deselected. The screenshot in the middle shows the scene with a rotated camera. The blue marked objects do not belong to the current scene part anymore. By selecting the manipulator on the viewing volume it can be modified. The result is shown in the right part of the image.

5.6 Result Images

The objects shown in the following images were modeled with Maya. The textures were created in Gimp. The images themselves are composed with theMPI-Builder.

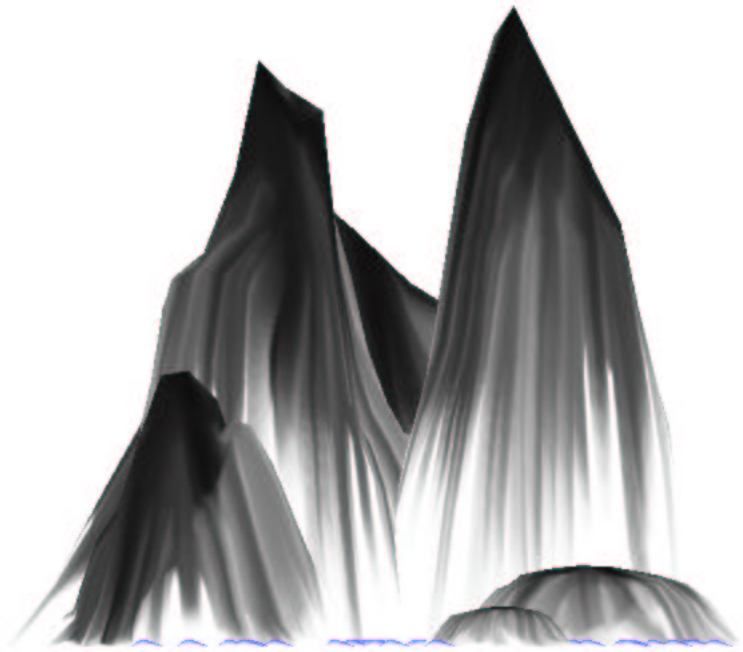

Figure 5.5: Parallel projected scene.

The first example presented here uses the most important elements in Chinese Landscape Painting, mountains and water. In Figure 5.5 the scene is displayed using a single parallel projection. The images in Figure 5.6 uses the typical division of the space in foreground, mid part, and background. The foreground is shown using an othographic projection slightly looking down, creating an effect of *level distance*. The mid part is displayed using an orthographic projection looking down from a remarkly higher position, creating an effect of *high distance*. This is brought to the extreme in the right image there the DOP is perpendicular to the ground plane. A perspective projection is used for the mountains in the background looking up to them. Thus an effect of *deep distance* is created.

In the second exampe the scene consists of two houses and a tree place on a ground plane with a hill behind them. Figure 5.7 shows the scene projected using a single perspective projection. In Figures 5.8 and 5.9 different projections and viewpoints were chosen for each object. As in Chinese Landscape Paintings the houses are displayed using oblique projections. Because different parameters were chosen, the lines of the houses point in different directions, creating the impression of different viewpoints. Figure 5.8 could be interpreted as standing between the houses and turning the head to the left and right showing the right side of the left house and the left side of the right house. Whereas

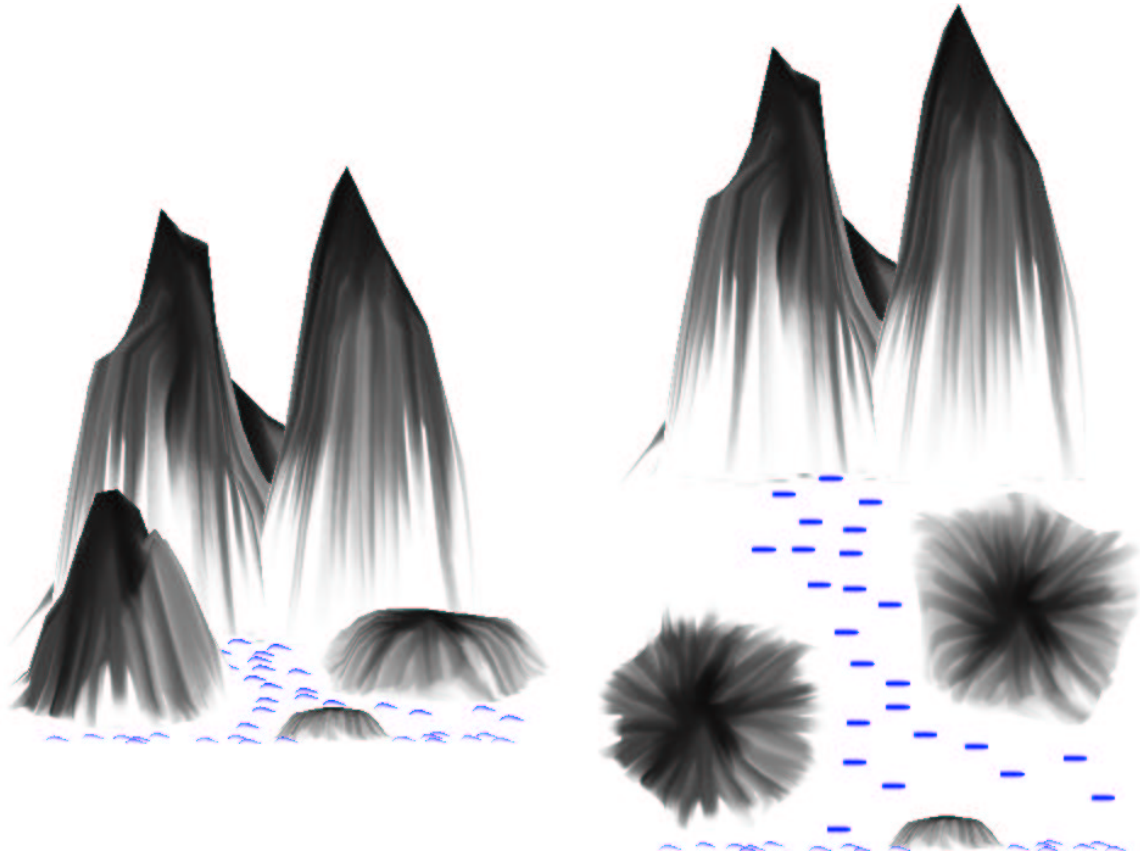

Figure 5.6: Multi projection images using different viewpoints for different parts of the scene. In the image on the right side an extreme viewpoint is used for the mid part.

the left house in Figure 5.9 would be seen from a position left of it, and the right house from a position far on the right side. The ground the houses are standing on is projected using an orthographic projection looking down from a slightly higher position. Thus the slope of the ground plane matches the slope of the houses. Furthermore looking down from a higher point shows much more of the ground. The tree and the hill are projected directly from the front using orthographic projections.

Figure 5.7: Single perspective projection.

Figure 5.8: Multi projection image using oblique orojections for architecture.

Figure 5.9: Using different parameters for the oblique projections.

Finally the images in Figure 5.10 show the difference between the *stacking* projection and object stacking. As the *stacking* projection puts points that are further away higher up in the image, it makes the top surface of objects more visible. This effect can be seen best for the hills in the lower right corner of the left image. Using the object stacking algorithm moves entire objects. The objects are displayed like using a single projection with the difference that additional vertical gaps are created between them. An orthogrphic projection was used (in combination with object stacking) to create the image on the right side. As in Figure 5.5 just the front surfaces of the objects are shown. But the stacking algorithm creates a distribution of the objects across the image. As only single objects that are not placed on top of each other are used in the model the object stacking algorithm produces pleasing results. Figure 5.11 illustrates the problems that can occur if a complex object is created using several other objects or if objects are placed on other objects. The houses are modeled using different objects, cylinders as the pillars and rectangles for the roof. For the front part of the house the pillars are a bit further away than the front edge of the roof. Thus they are put up higher than the roof, so that they appear to break through it. The back part of the roof is also put up higher than the front part, so that they are not connected anymore. This problem might me solved on the model base, if the house object would be marked as an object composed of several other objects. It then could be treated as a single object by the algorithm.

Another effect resulting from the stacking algorithm is, that objects which are placed on top of other objects appear flying above them. Like the houses and the tree in Figure 5.11, that were originally placed on the ground plane.

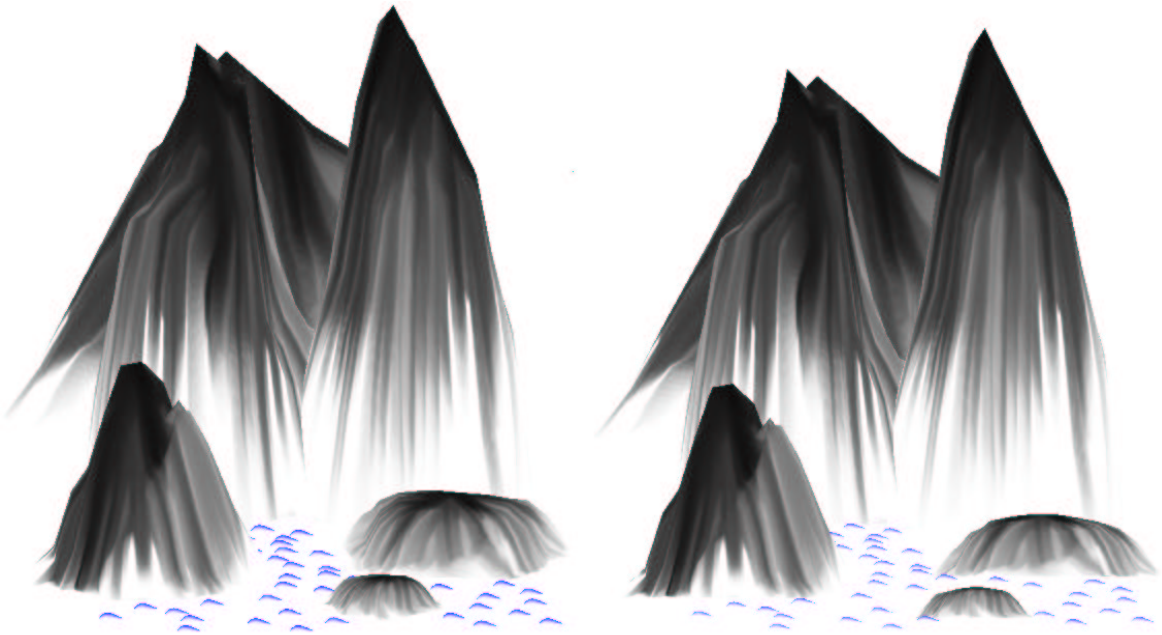

Figure 5.10: The *stacking* projection shows the top surfaces of objects (left). Object stacking algorithm moves entire objects (right).

Figure 5.11: Problems that can occur by using object stacking.

Chapter 6

Conclusion and Future Work

Alternative illustration methods become more and more significant. While the use of a single perspective projection allows the creation of photo-realistic images, it also can be restrictive. To overcome the restrictions multiple projections and view points in a single image can be used. In fine arts such as Chinese Landscape Painting such techniques are known. As exposed in the introduction multi projection images can be used for artistic expression, representation, and visualization.

In this thesis an alternative rendering pipeline for the creation of multi projection images was introduced. It is based on the composition of Chinese Landscape Paintings. The rendering pipeline was implemented in an application named MPI-Builder. The application provides a set of planar geometric projections, an user interface for the segmentation of the scene into scene parts, controlling camera settings and the projections. If a user manipulates the values of the projection visual feedback is given. Obviously the resulting image changes, but the user can also see how the viewing volume of the projection is affected. Furthermore most parameters of the implemented projections can be adjusted interactively by manipulating the viewing volume. The values of the parameters are updated accordingly. This kind of feedback gives the user a good understanding of how the parameters of a projection influence the resulting image.

Future Work

The user interface could not be tested in detail yet. There are certainly functions which could be added, such as a full rotatable view onto the scene in the Scene View Window, the ability to connect parameters such as the scaling parameters of a projection to allow a uniform scaling by changing only one value, and more. Further tools or dependencies which would help the user to compose a multi projection image are imaginable. The object stacking algorithm leaves unsolved problems as well, which might be solved on the model base or in the application.

Beside these possible approvements of the existing work, further development in the following fields is imaginable:

- At this point work has been done in simulating eastern perspective and in simulating Chinese drawing styles separately. Combining those in one application would allow to create images that are much more like the original Chinese Landscape Paintings.
- In this work the colors of objects were predetermined by textures, only. Integrating light sources and dealing with shadows and reflections in multi projection images is another future direction. Especially the transition from one layer to another would be interesting.
- Finally the question, if animation in terms of navigating through a multi projection scene would be possible, is interesting. Problems of automatic scene part generation and management, and frame-to-frame coherence needs to be solved then.

Bibliography

[AZM00] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic Multiprojection Rendering. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 125–136, June 2000.

[Cah77] James Cahill. *Chinese Painting*. Skira Rizzoli, New York, 1977.

[Chi04] China the Beautiful. Web page: http://www.chinapage.com/, Januar 2004.

[dS68] Anil de Silva. *The Art of Chinese Landscape Painting*. Greystone Press, New York, Toronto, London, revised edition, 1968.

[FCL03] Farzam Farbiz, Adrian David Cheok, and Paul Lincoln. Automatic Asian Art: Computers Converting Photos to Asian Paintings using Humanistic Fuzzy Logic Rules. In *SIGGRAPH*, San Diego, CA, USA, 2003.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: principles and practice*. Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1990.

[HH91] David Hockney and Philip Haas. A Day on the Grand Canal with the Emperor of China. Or Surface is Illusion but so is Depth. Milestone Film & Video, 1991.

[Int98] Interplay. Fallout 2. Web page: http://www.interplay.com/fallout2/, 1998.

[MCMW01] Kaye Mason, Sheelagh Carpendale, Peter MacMurchy, and Brian Wyvill. Eastern Perspectives and the Question of Personal Expression. Technical report, University of Calgary, 2001.

[Rob97] Nate Robins. Opengl Tutors.
Web page: http://www.xmission.com/ nate/tutors.html, 1997.

[Sin02] Karan Singh. A Fresh Perspective. In *Proceedings of the Graphics Interface 2002*, pages 17–24, Mississauga, Ontario, Canada, 2002. Canadian Information Processing Society.

[Sta94] Wolf Stadler. *Lexikon der Kunst in zwölf Bänden.* Karl Müller Verlag Erlangen, Erlangen, Germany, 1994.

[Tro04] Trolltech. Creators of Qt. Web page: http://www.trolltech.com/, 2004.

[WLS02] Der-Lor Way, Yu-Ru Lin, and Zen-Chung Shih. The Synthesis of Trees in Chinese Landscape Painting using Silhouette and Texture Strokes. In *Journal of WSCG*, pages 499–506, Pilsen, Czech Republic, 2002. UNION Agency - Science Press.

[WNDS99] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide.* Addison-Wesley Publishing Company, Reading, MA, 3rd edition, 1999.

[WS01] Der-Lor Way and Zen-Chung Shih. The Synthesis of Rock Textures in Chinese Landscape Painting. In *Proceedings of the 22nd Annual Conference of the European Association for Computer Graphics*, volume 20, 3, pages 123–131, Oxford, UK, 2001. Blackwell Publishers.

Independence Statement

Herewith I declare that I have completed this work solely and with only the help of the mentioned references.

Magdeburg, 31st May 2004