Constraint-Based Rendering for Scenes with High Dynamic Ranges

by

Lijiang Fang

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1993

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Many researchers have examined rendering techniques with a focus on realistic image synthesis. Ray tracing and radiosity, which are the most successful current methodologies, are based on the physics of light and surfaces. Neither considers display device limitations or properties of human visual perception. Furthermore, the synthetic camera model has shown its deficiency in rendering images with high dynamic ranges onto display devices with lower dynamic ranges.

A new rendering framework is proposed. Human visual properties are incorporated into the framework to increase the effective visual contrast. It is known in visual perception that brightness is not a monotonic function of intensity. The perceived brightness is affected by the intensities of the surrounding area. It is also known that human vision is insensitive to low frequency spatial intensity variation. In the proposed framework, to preserve the visual contrast in one image, the contrasts across edges are maintained while the intensities in large areas are slowly varied.

Based on the proposed framework, a modified rendering pipeline is presented and a prototype system is implemented. The system generates the contrast constraints by invoking a modified visible surface algorithm. Then, the problem of satisfying the constraint hierarchy is transformed into a bounded linear least squares (BLLS) problem. Numerical algorithms are employed to solve the BLLS problem.

Acknowledgements

First, I would like to thank my supervisor Dr. William Cowan for his guidance, encouragement, and support. He also suggested I improve my English writing even before he became my supervisor, so that the writing of this thesis was less painful. I would also like to thank my faculty readers, Dr. Richard Bartels and Dr. Anna Lubiw, for spending time reading my thesis and providing helpful suggestions and comments.

I enjoyed the stimulating environment in the Computer Graphics Laboratory (CGL). I thank all the faculty members, staffs, and students of CGL. In particular, I would like to thank Eric Davies, for serving as my thesis's student reader and for many discussions on graphics, X windows, LaTeX, etc. Also thanks to Lori Case of the Scientific Computation Group for discussing numerical optimization with me; to Maureen Stone of Xerox PARC for making a manuscript available to us; to Steve Mann for making ice cream weekly in CGL. Thanks also go to Rick Stroobosscher, Phil Bertrand, Peter Mayo, and Rob Kroeger for providing the friendly computing environment in CGL.

Thanks to all my friends for making my stay at Waterloo enjoyable. Special thanks to my family members for their moral support. My brother Liping introduced me to Waterloo and helped me in many ways while I was far away from my parents.

This work was partially supported by an Ontario Graduate Scholarship; by several ITRC fellowships; by NSERC and ITRC through various grants to CGL; and by Digital Equipment Corporation through equipment donations to CGL.

Trademarks

DECstation and ULTRIX are trademarks of Digital Equipment Corporation. X Window System is a trademark of the Massachusetts Institute of Technology. NAG is a registered trademark of the Numerical Algorithms Group Limited, the Numerical Algorithms Group Inc, and the Numerical Algorithms Group (Deutschland) GmbH.

All other products mentioned in this thesis are trademarks of their respective companies. The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not indentified, is not to be taken as a sign that such names may be used freely by anyone.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Rendering in the Context of Computer Graphics

Two major areas of research in computer graphics are modelling and rendering. Research on modelling attempts to provide easy and powerful ways for a user to specify and manipulate a scene. Rendering refers to the procedure of creating a visible image on display hardware from the input model. It can be viewed as a function which maps a scene description to pixel values on specific display hardware. The rendering procedure can be divided into several stages: the modelling transformation, the viewing transformation, projection, clipping, window to view-port mapping, scan conversion, and digital to analog conversion (DAC) for the display hardware (e.g. shadow-masked cathode ray tube (CRT) controlled by the voltage). These stages constitute the rendering pipeline. The objective of research on rendering is to improve the image quality and the computational efficiency of rendering algorithms (the mapping functions). As reviewed in Chapter 2, many researchers have examined rendering techniques with a focus on real-

istic image synthesis. Ray tracing and radiosity are the most successful current rendering methodologies.

1.2 Display Device Limitation

There are many different display devices, such as the CRT, liquid-crystal display (LCD), plasma display, electrostatic printer, and thermal-transfer printer. In this thesis, display device refers to the CRT because it is the most popular display medium in interactive computer graphics, but the techniques apply to all devices.

A CRT is not able to emit a radiance smaller than a minimal radiance or bigger than a maximal radiance. These restrictions cause many problems in computer graphics. In this thesis, we only focus on the fact that it is impossible for a CRT to display absolute dark as discussed in detail in Chapter 2. Because of this limitation, the contrast available on a CRT is also limited. If we want to render a scene whose contrast is much higher than the attainable contrast on a CRT using straightforward rasterization techniques, the scene sustains a loss of information (Chapter 4 will discuss this problem in detail).

1.3 Motivative Examples

Some examples in photography and painting inspire our interest on addressing the problem of display device limitation in computer graphics. Let us describe a scene that we see quite often in daily life. A bright sun in the sky illuminates a tree from behind. We would like to take a picture of this scene. Because we are looking in shadow, if we focus our camera at the tree, we get a photograph in

which the sun is not as bright as it should be and does not look too much different from the surrounding sky. On the other hand, if we focus the camera at the sun, although the sun is very bright, but the tree looks extremely dark with little or no visible detail. The reason why we have this problem is that the contrast of the scene is far out of the range of the available contrast on the photograph.

In his *The Empire of Lights* ([Gim87, illustrations 61-65] and [Sob65]), Magritte painted a picture which consists of a dark night scene with a light shining on a house and its surrounding trees, etc., together with a bright day sky. The original brightness contrast of these two components are far out of the range which a painting could deliver. However, Magritte skillfully varied the luminances along the picture, and we have a feeling that the sky is as bright as the day sky and the house and its surrounding are in the night, and we still can see the details of the sky and the details of the house and its surroundings. We did a measurement using a radiometer on the luminance distribution with this painting. We found, for instance, that in one tree, some parts which look at the same brightness actually have different luminances. We also found that the leaves bordering with the sky have lower luminances, resulting in a high contrast which makes the sky look brighter.

From these two examples, we can see that the synthetic camera model in graphics rendering has its weakness with high contrast images, and the artistic approach may provide better results.

1.4 Objective of the Research

From the discussion in Sections 1.1-1.3, we see that the display hardware in computer graphics restricts the contrast attainable in the image, and the synthetic camera rendering model cannot cope with this deficiency. In this thesis, we examine the restriction on display devices in depth, and propose a new framework for rendering that takes into account properties of the human visual system. To the human visual system, the absolute luminance of an object is less important than comparing the luminances of neighbouring objects (i.e. local contrast). In the proposed framework, we will use a constraint-based approach, which considers the relationship among neighbouring objects, and attempts to maintain the contrast at some places while relaxing the contrast at other places depending on the relationships.

1.5 Related Work

In the literature, there has been little work done on rendering of high dynamic range images with the consideration of display device limitation and human visual properties. However, the following work is related to our research.

Klassen investigated the problems of device dependent image construction [Kla89]. He examined faithful image reconstruction that takes into account spatial and temporal properties of individual pixels (pixel structure artifacts). The pixel intensity profile for a CRT is Gaussian distribution in space and exponential decay over time. The problem discussed in his work is in fact different from the one considered in this thesis. His research sought to reduce or eliminate the pixel structure artifacts with given image intensity distribution and to minimize

the difference between the given image and the image displayed. The problem considered here is to reduce the information loss during the rasterization. In our research, when a high contrast image is rendered, some of its intensities may be altered in order to preserve the contrast at important locations.

Tumblin and Rushmeier addressed the drawback of simple normalization in realistic computer generated images [TR91]. Because the simple normalization just divides all the radiance values, which are computed by radiosity or ray tracing methods, by the strongest radiance in an image, a firefly-powered image and a searchlight-powered image become undistinguishable. They applied the Stevens and Stevens models of brightness versus luminance relations [SS60, SS63] to create a tone reproduction operator for black and white computer generated images. As a result, dark scenes look dark and bright scenes look bright. Their work is related to this thesis in the sense that it also considers display device restrictions and properties of the human visual system, but its focus is different from ours.

Chiu et al. presented an algorithm that spatially nonuniformly scales the pixel intensities of a computer generated grey-scale image so that the intensities are all displayable on a CRT [CHS+93]. The scaling function is constructed from a low pass filter. For the same intensity at different pixels, the function value is larger at a pixel whose neighbouring pixels have lower intensities and is smaller at a pixel whose neighbours have higher intensities. The scaling function is built by ad hoc techniques, and thus it is probably image dependent. Because their method works in the raster space and it does not know the internal object structure, it is hard for the algorithm to take advantage of the object structure and it may create artifacts.

Glassner et al. proposed a device-directed rendering methodology [GFMS93].

They represent the image by a symbolic representation that describes the relationship between the scene lights and surfaces and the pixel colours. If there exist some pixel colours out of the device gamut in the image, the image is projected onto the device gamut. The projected image is called the target image. A minimization procedure is then performed to adjust the light and surface parameters so that the resulting image has minimum distance from the target image. If the resulting image still has pixel colours out of the device gamut, a new iteration is carried out again. Their methodology strictly follows the physics of lights and surfaces. It has the advantage of automatically adjusting the scene so that the image is guaranteed to be in device gamut without rendering artifacts caused by gamut-mapping. However, the method alters the scene lights and surfaces, which may cause the problem of moving the image farther from a modeller's intention, and the method does not appeal to the properties of visual perception to achieve visual effect beyond the physical limitation of a given device.

1.6 Outline of the Thesis

Chapter 2 introduces the background on rendering, including the physics of light and surfaces, approximate lighting models, shading models, ray tracing, radiosity, and the working mechanism of CRTs. The background on visual perception is discussed in Chapter 3, emphasizing properties of spatial vision. We propose a constraint-based rendering approach in Chapter 4, which incorporates properties of visual perception into rendering and considers the display device limitation. The implementation details are explained in Chapter 5. The modified rendering pipeline is described with the emphasis being on constraint generation and constraint solving. Finally, we summarize the thesis and point out possible

future work in Chapter 6.

Chapter 2

Background on Rendering

Rendering is the process of creating images from models [FvDFH90, page 606]. In frame buffer architecture, rendering is also referred to as rasterization for the procedure of computing the pixel values from the input graphics models. This chapter will review the physics foundation behind rendering and the work done by other researchers on rendering techniques. We will find that these researchers do not pay any attention to limitations of the display device (e.g. CRT) and to the properties of the human visual system.

2.1 Physics of Lights and Surfaces, Lighting Models

The ultimate objective of realistic image synthesis is to produce an image as close to perceived reality as possible. It does so by simulating the interactions between lights and surfaces within an environment. However, these simulations are very computationally expensive, or even untractable. To reduce the computational complexity, many approximation models have been proposed. We first

briefly introduce the physics of lights and surfaces, and then give the popular approximate lighting models.

2.1.1 Physics of Lights and Surfaces

A good introduction to the physical process of light and colour can be found in [CSW87] or [JW75]. Light rays start from an illuminant (lighting source), fall on the object, then reflect and / or scatter to the viewer's eyes. From modern physics, we know that light is made of particles called photons. A photon is characterized by energy E, frequency ν, wavelength λ, and wavenumber $\nu\prime$. These quantities have the relationships as in Equations 2.1, 2.2, and 2.3.

$$E = h\nu \tag{2.1}$$

$$\lambda\nu = c \tag{2.2}$$

$$\nu\prime = \frac{1}{\lambda} \tag{2.3}$$

where $h = 6.62 \times 10^{-34}\, joule \bullet seconds$ and $c = 3.00 \times 10^{8} metres/second$.

An illuminant is specified by its spectral power distribution Φ_λ. $\Phi_\lambda \Delta\lambda$ describes the number of photons within the wavelength interval $[\lambda, \lambda+\Delta\lambda]$. Photons of wavelength below $480nm$ appear blue, photons of wavelength around $520nm$ appear green, and photons of wavelength around $580nm$ appear yellow, while photons of wavelength above $600nm$ appear red. When an illuminant emits more photons of long wavelength, it looks reddish, and when it emits more photons of short wavelength, it looks bluish.

When the photons from an illuminant reach the surface of an object, the surface re-emits them in two different ways. Some of the photons are reflected with

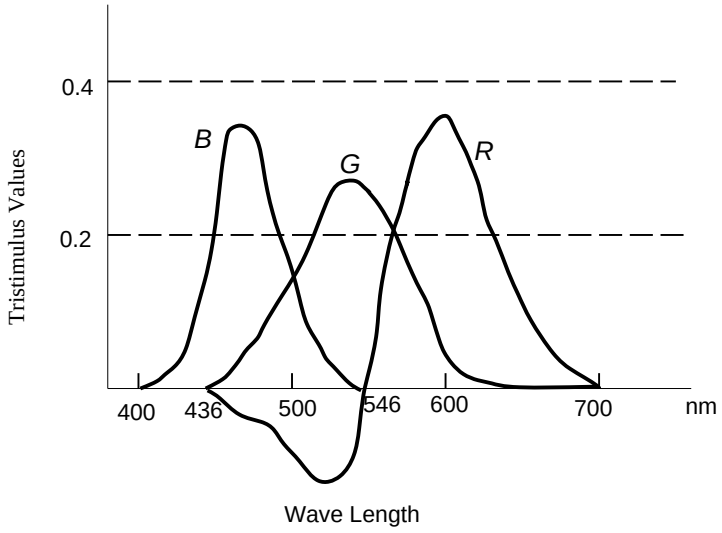
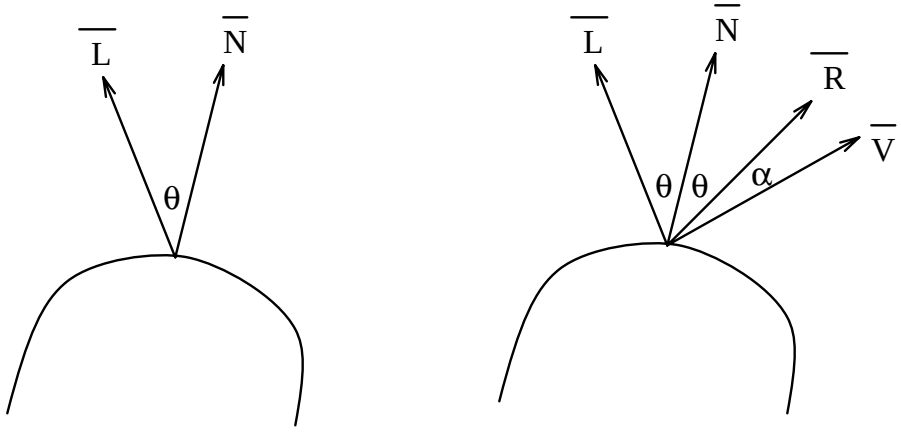


Figure 2.1: Colour Mixture Functions for Red, Green, and Blue Primaries for the Visible Spectrum (Primaries are at 436, 546, and 700nm respectively) (adapted from [CSW87])

the angles of reflection nearly equal to the angle of incidence, which is called specular reflection. Some photons are scattered. However, the number of photons falling on a surface is not equal to the number of photons leaving the surface. The number of photons leaving the surface with wavelengths between λ and $\lambda + \Delta\lambda$ is $R(\lambda)\Phi_\lambda\Delta\lambda$, where $R(\lambda)$ is the reflectance of the surface for wavelength λ, which satisfies $0.0 \leq R(\lambda) \leq 1.0$ assuming no fluorescence. Effectively the surface subtracts $(1 - R(\lambda))\Phi_\lambda\Delta\lambda$ photons from the arriving photons. This is the subtractive colour model. The light seen by an observer is determined by the reflectance function of the surface and the spectral distribution of the illuminants.

Precise spectral representation of lights is extraordinarily inefficient. In computer graphics, lights are usually represented by a mixture of **red, green**, and



(a) Diffuse Reflection (b) Specular Reflection

Figure 2.2: Diffuse and Specular Reflections

blue primaries. Figure 2.1[1] shows that most of the colours in the visible spectrum can be simulated by adding amounts of red, green, and blue, although the colour whose wavelength is in the interval $[436nm, 546nm]$ can not be additively simulated.

2.1.2 Lighting Models

[FvDFH90, pages 722-734] has a good introduction to lighting models. There are two types of lights. One type is the light bathing a surface from the surroundings. The other type is that due to light sources. They are the **ambient light** and the **direct light** from an illuminant. The light re-emitted from a surface is in the form of **diffuse light** and **specular light**. Ambient lighting is the light which constantly illuminates all the objects in an environment. For a given surface, the intensity is usually taken to be a constant. Diffuse lighting is the light reflected with equal

[1] All the figures in this thesis are only to illustrate some concepts or trends. They are not necessary quantitatively accurate.

intensity in all directions. For a given surface, the intensity of diffuse reflection only depends on the angle θ between the normal $\overline{N}$ of the surface and the incident light direction $\overline{L}$. Specular lighting is the light reflected unevenly in different directions. It only reflects in directions aligned with, or nearly with, the direction of reflection $\overline{R}$. $\overline{R}$ can be calculated by $\overline{R} = 2\overline{N}(\overline{N} \bullet \overline{L}) - \overline{L}$. Figure 2.2 illustrates diffuse and specular reflections. In the figure, $\overline{V}$ is the viewing direction. The illumination at one point of a particular surface can be calculated by Equations 2.4, 2.5, and 2.6.

$$\begin{aligned} I_R &= S_{a_R} \times I_{a_R} + \\ &\quad S_{d_R} \times I_{S_R} \times (\overline{L} \bullet \overline{N}) + \\ &\quad S_{s_R} \times I_{S_R} \times (\overline{R} \bullet \overline{V})^n \qquad (2.4) \\ I_G &= S_{a_G} \times I_{a_G} + \\ &\quad S_{d_G} \times I_{S_G} \times (\overline{L} \bullet \overline{N}) + \\ &\quad S_{s_G} \times I_{S_G} \times (\overline{R} \bullet \overline{V})^n \qquad (2.5) \\ I_B &= S_{a_B} \times I_{a_B} + \\ &\quad S_{d_B} \times I_{S_B} \times (\overline{L} \bullet \overline{N}) + \\ &\quad S_{s_B} \times I_{S_B} \times (\overline{R} \bullet \overline{V})^n \qquad (2.6) \end{aligned}$$

where I_R, I_G, and I_B are the intensities for R, G, and B, S_{a_R}, S_{a_G}, and S_{a_B} are the surface ambient coefficients, I_{a_R}, I_{a_G}, and I_{a_B} are the environmental ambiences, S_{d_R}, S_{d_G}, and S_{d_B} are the diffuse coefficients, I_{S_R}, I_{S_G}, and I_{S_B} are the intensities of the lighting source, S_{s_R}, S_{s_G}, and S_{s_B} are specular coefficients, n is a constant depending on the property of the surface, and $\overline{L}$, $\overline{R}$, $\overline{N}$, and $\overline{V}$ are the same as in Figure 2.2.

2.2 Shading Models

To **shade** is to paint a surface with an appropriate colour distribution. A very detailed introduction on shading can be found in [FvDFH90, pages 734–739]. Its objective is to display surfaces as close to the surfaces in an ideal camera-shot photograph as possible. During the last two decades, many shading models have been proposed. The most frequently used shading models are flat shading, Gouraud shading, and Phong shading. They vary in image quality and computational cost. The first approach has the cheapest computing cost and worst image quality, while the third has the most expensive computational cost and highest image quality.

2.2.1 Flat Shading

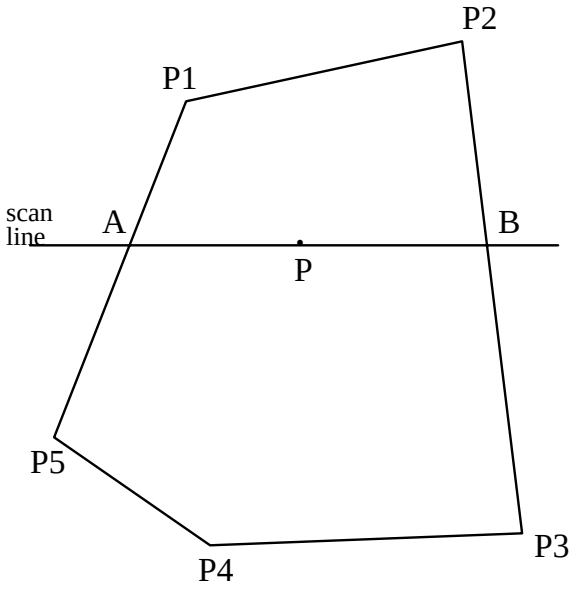
It was assumed that the shading of one surface is uniformly distributed, i.e. every point on one surface has the same properties. Given the surface properties, the ambient and diffuse coefficients, we can calculate the shade of the surface with following formulaes:

$$I_R = S_{a_R} \times I_{a_R} + S_{d_R} \times I_{S_R} \times (\overline{L} \bullet \overline{N}) \quad (2.7)$$

$$I_G = S_{a_G} \times I_{a_G} + S_{d_G} \times I_{S_G} \times (\overline{L} \bullet \overline{N}) \quad (2.8)$$

$$I_B = S_{a_B} \times I_{a_B} + S_{d_B} \times I_{S_B} \times (\overline{L} \bullet \overline{N}) \quad (2.9)$$

During the scan conversion, we use the same R, G, and B values for every pixel within this surface. The result is a set of constant coloured polygons, hence the term **flat shading** is used.



Assume:

$A = sP1 + (1-s)P5$
$B = tP2 + (1-t)P3$
$P = rA + (1-r)B$

We have:

$I_A = sI1 + (1-s)I5$

$I_B = tI2 + (1-t)I3$

$I_P = rI_A + (1-r)I_B$

Figure 2.3: Gouraud shading

2.2.2 Gouraud Shading

Gouraud shading ([Gou71] and [FvDFH90, pages 736-737]) is one of the most commonly used shading methods. It first computes every polygon normal, then it computes the normal of a vertex by the average of the normals of all the polygons which connect to this vertex. Assuming that the optical properties for all the polygons in one object are the same, after having the normal for a vertex, it may compute the shading for this vertex by Equations 2.4, 2.5, and 2.6 . It computes the shadings for all the vertices in this way, then it calculates the shading value for a point on the edge by linearly interpolating between two end vertices. The shading value for any pixel can be calculated by linearly interpolating the values of two points on the polygon edges across scan line. Figure 2.3 illustrates this procedure.

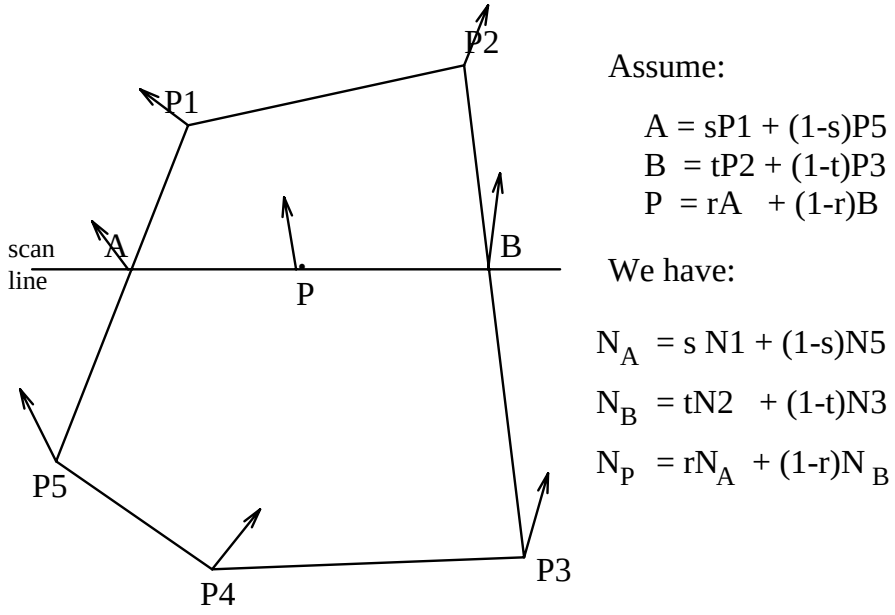


Figure 2.4: Phong shading

2.2.3 Phong Shading

Phong shading ([BT75] and [FvDFH90, pages 738-739]) is another common shading method. It has higher image quality than Gouraud shading. We can also see that Phong shading is more flexible than Gouraud shading because Gouraud shading has to assume all the polygons in one object have same surface property. This process is shown in Figure 2.4 .

We see that flat shading only computes the shade once for one surface, Gouraud shading computes the shade at each vertex and the other pixels use the linear interpolation of shades for those vertices, and Phong shading computes the shade at every pixel by the interpolation of the normals of the vertices. Therefore, Phong shading is the most expensive and flat shading is the cheapest.

The aforementioned shading techniques are computationally efficient with

reasonable image quality. However, if we wish to have highly realistic images, these methodologies are not sufficient. For example, shadows can not be easily accommodated using these models, and interpolation of either normal or shading values is an approximation. To provide better photo-realistic image synthesis, two different approaches have been taken. One is ray tracing [FvDFH90, pages701-715, 753, 776-793, 804-806] [Gla89], and the other one is radiosity [FvDFH90, pages722, 753, 775, 793-806] [GTGB84].

2.3 Ray Tracing

A simple **ray tracer** casts a viewing ray from the eye point to every pixel on the screen. If a viewing ray from the eye point to one pixel intersects object surfaces, only the nearest surface is visible. The colour on this pixel is calculated based on the lighting properties of this surface. The ray tracer casts lighting rays to all the light sources from the intersection point of the viewing ray and the nearest object. If a lighting ray does not intersect any object, then this light contributes diffuse and specular illumination for that pixel. If the object is reflective, the ray tracer then recursively fires a reflection ray in the reflection direction of the viewing ray from the eye point, and the intensities computed from recursive ray are discounted by a factor and added to the intensity values for the viewing ray. If the object is refractively transparent, the ray tracer recursively fires a refraction ray through the object in an appropriate direction and applies the recursively computed intensity, discounted by a factor, to the viewing ray.

From [FvDFH90], we find that recent trends in ray tracing research divide into two categories. They are to improve the efficiency of ray tracing and to improve the image quality. Item buffers [WHG84], reflection maps [Hal86],

adaptive tree-depth control [HG83], light buffers [HG86], and ray classification [AK87] have been proposed to speed up ray tracing. Cone tracing [Ama84], beam tracing [HH84], pencil tracing [STN87], stochastic sampling [CPC84, Coo86], and path tracing [Kaj86] have been used to enhance the image qualities of ray traced images.

2.4 Radiosity

Radiosity is a word borrowed from illumination engineering. Although ray tracing works well to model specular reflection and dispersionless refractive transparency, it still uses the ambient light to measure the other global lighting contributions. Assuming the conservation of light energy in a closed environment, radiosity treats inter-object reflection more accurately. Suppose that the environment could be broken into n discrete patches, radiosity assumes that the light energy leaving a unit area is equal to the energy emitted in unit area from that patch and the energy reflected in unit area from that patch as shown in Equation 2.10.

$$B_i = E_i + \rho_i \sum_{1 \le j \le n} B_j \frac{F_{j-i} A_j}{A_i} \tag{2.10}$$

B_i and B_j are the radiosities of patches i and j, measured in $energy/unit\,time/unit\,area$. E_i is the rate at which light energy is emitted from patch i, measured in the same unit as radiosity. ρ_i is the reflectivity of patch i. A_i and A_j are the areas for patches i and j. F_{j-i} is a form factor, a value that specifies the percentage of energy leaving from patch j that reaches patch i.

A_i, E_i, and ρ_i are the properties of a patch i which are known. F_{j-i} can be computed according to patch i and patch j's geometries and their relative locations [FvDFH90, pages795-799]. Therefore, only B_i and B_j are unknown in Equation 2.10. By solving the system of Equations 2.10 for i from 0 to n, the radiosity for each patch is computed. It should be noted from Equation 2.10 that the radiosities of patches i and j do not contain any information about the viewpoint. Radiosity is a view independent property of a surface.

According to [FvDFH90], current research in radiosity focuses on the efficiency of radiosity and the image qualities of radiosity generated images. Cohen and Greenberg used a hemicube to improve the efficiency of approximating the form factor [CG85]. Cohen et al. adaptively subdivided subpatches at places of high radiosity gradient to reduce the number of patches and still preserve the image quality [CGIB86]. Immel et al. extended the radiosity method to model specular reflection [ICG86].

Radiosity is a view-independent image synthesis technique, which has advantages for calculating the diffuse component of illumination, while ray tracing is a view-dependent image synthesis technique, which is better in calculating the specular component. Wallace et al. implemented a two-pass approach to simulate global illumination within complex environments [WCG87].

2.5 How CRTs Work

After the pixel values are computed by the chosen rendering models, the next step is to place those calculated values onto a display surface. There exist many display devices available for computer graphics. They include shadow-masked

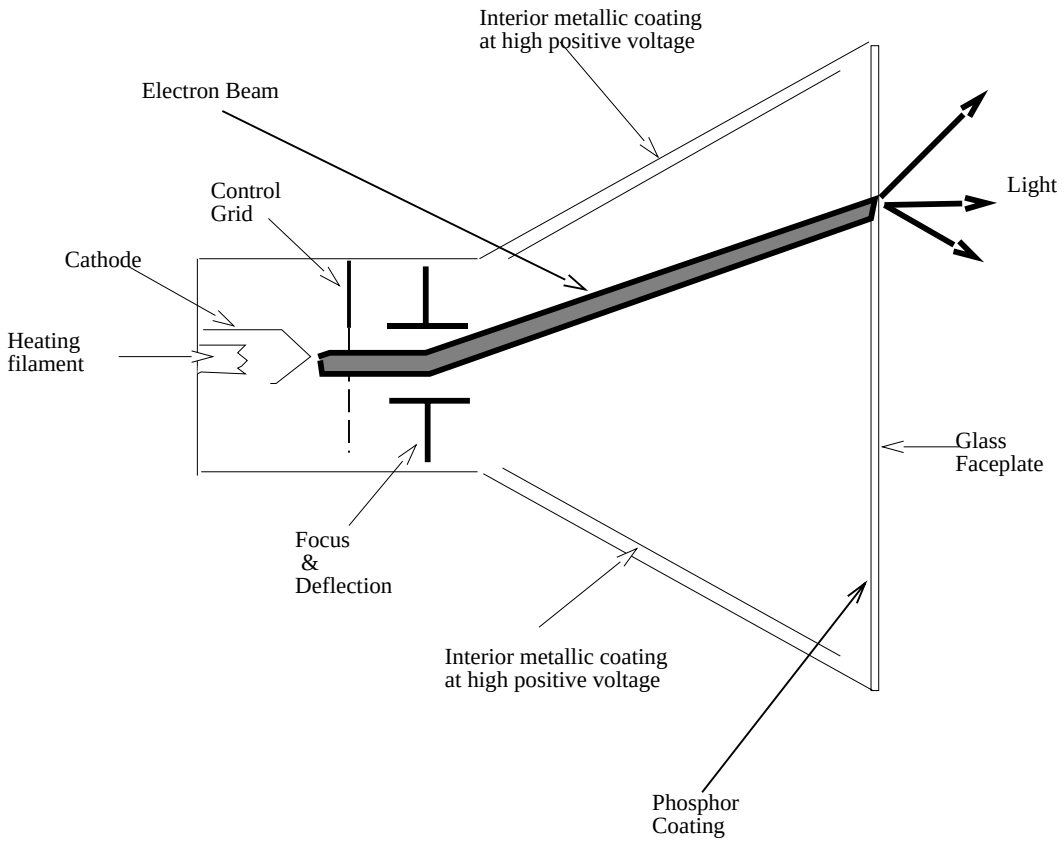


Figure 2.5: Schematic Diagram of A Monochrome CRT (adapted from [Cow87])

cathode ray tubes (CRT), direct-view storage tubes (DVST), laser printers, pen plotters, and liquid-crystal displays. Currently, the most frequently used display hardware in interactive computer graphics is the CRT. Therefore, this thesis selects CRTs to explore the display contrast limitation and its impact on rendering.

2.5.1 CRT Structure

Figure 2.5 is the cross-section of a CRT. The main components of a CRT are labeled in that figure. The electron gun in a CRT consists of a heating filament and a

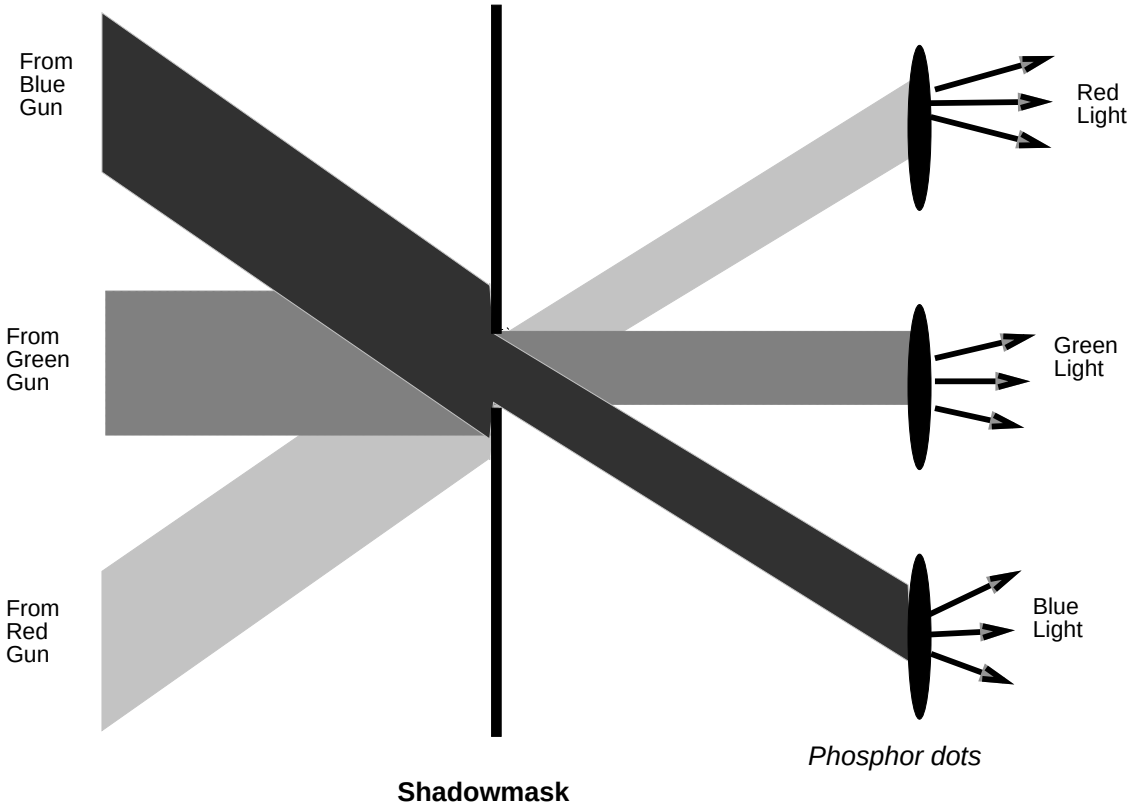


Figure 2.6: Shadow-Mask (adapted from [Cow87])

cathode. The heating filament heats the cathode causing it to emit electrons. Some of these electrons pass through the control grid which determines how many electrons are actually in the beam. This determines the brightness of the spot on the screen where this beam hits. This beam is accelerated by the interior metallic coating near the screen which has high positive voltage charge. The focusing system forces the electrons into a narrow stream. This stream is directed toward a specific point on the screen by the magnetic field generated by the deflection coils. When the electron beam hits a point on the screen, the phosphor on the screen emits light.

The previous paragraph only describes the principles of a monochromatic CRT. A colour CRT has a similar operation as a monochromatic CRT. It has three electron guns, one each for red, green, and blue respectively. A shadow-mask is mounted behind the screen and each pixel contains three closely grouped phosphor dots for red, green, and blue. The colour a viewer sees on a pixel is the mixture of the colours of these three dots. This colour can be controlled by alternating the strength of red, green, or blue. The shadow-mask has one hole for each pixel. It is precisely aligned so that only one electron gun is on the line formed by this hole and one phosphor dot. Hence, a phosphor dot is only exposed to the electrons shot from a specific gun. Two type of shadow-mask CRTs exist. They are the *delta-delta* CRT, in which both the phosphor dots and the electron guns are aligned in triangle, and the *precision in-line delta* CRT, in which both the three guns and the phosphor dots for one pixel are located in one line instead of in a triangle. An electron beam in the latter is easier to converge to the destined phosphor dot than that in the former but the latter slightly reduces image sharpness. Figure 2.6 illustrates the geometrical principle for the shadow-mask in a shadow-mask CRT.

2.5.2 Gamma Correction

The pixel intensity (I) is determined by the number of electrons (N) exposed to the phosphor dots for that pixel. However, this relationship is not a simple linear function. Instead we have the equation $I = kN^{\gamma}$, where k and γ are constants for the set up of a particular CRT. If the adjustments of CRT controls is unknown, most CRTs have γ between 1.4 and 4 [Mac91, MC92]. If a CRT is properly set up, the typical γ value is around 2.3. The set up of a CRT will be discussed in the

next subsection.

The number of electrons in the beam is proportional to the control-grid voltage. Hence, the control-grid voltage and the intensity for a pixel have the following correspondence $I = KV^{\gamma}$, where K is a new constant for the given CRT setting. This is equivalent to $V = (\frac{I}{K})^{\frac{1}{\gamma}}$. Therefore, in order to have intensity I on a pixel, we should adjust the control-grid voltage to $V = CI^{\frac{1}{\gamma}}$, where $C = (\frac{1}{K})^{\frac{1}{\gamma}}$.

2.5.3 CRT Set Up and CRT Contrast Limitation

It is impossible for a CRT to display absolute zero intensity because light reflects from the phosphor within the CRT and ambient light scatters on the display surface. If we denote the maximum intensity as 1.0, the minimum attainable intensity is from $\frac{1}{200}$ to $\frac{1}{40}$ for CRTs [FvDFH90, page 564]. The ratio of the maximum and minimum intensities is called the **dynamic range**. The dynamic range of a CRT is strongly affected by the lighting condition in the room where the CRT is located. However, if we set up a CRT properly, the dynamic range can be maintained at a perceptually optimal level. This subsection describes a standard CRT setup method [Mac91, MC92].

A CRT usually has several controls, such as brightness, contrast, focus, underscan / overscan, pedestal, gain, and horizontal / vertical size, to adjust the various image properties [Cow87, pages 13-15]. The controls that are of importance to the luminance limit of a CRT are brightness, contrast, and focus controls. The brightness knob adjusts the background level (black level) of light on the screen and also varies the aforementioned γ. γ becomes bigger as the background level is increased. The contrast control changes the ratio between the intensities of

the brightest possible value and the darkest possible value. The focus control modifies the size of the electron beam. The optimal beam size is just large enough so that there is no dark grid on a uniform field.

The standard CRT set up to adapt to the lighting condition of a room is as follows. First, the focus is set by the *shrinking-raster* method: a flat field is displayed, and we adjust the focus knob so that the image is as sharp as possible while the raster lines are still invisible. Second, to set brightness (black level), we vary the brightness knob to the brightest level consistent to zero input producing perceived black under the usual ambient illumination conditions. Finally, the contrast is set by adjusting the contrast level to the maximum at which there is no blooming when the CRT is used at full intensity input. Blooming is the phenomenon that the electron guns supply too much energy to the screen. This results in wide, blurry pixels and desaturated colours. Because brightness and contrast are adjusted separately, iterations may be required.

Chapter 3

Background on Visual Perception

This chapter offers a tutorial on human visual perception to audiences with a computer graphics background. Most of the materials to be presented in this chapter can be found in [Cor70, JW75, CSW87, FLS77]. All or at least most of the images generated in computer graphics are to be viewed by human beings regardless of the medium on which the images are produced. Therefore, knowledge of the principles of human visual perception is essential for producing images with good visual effects. In this chapter, we first explain how human beings perceive a scene, then we discuss some visual properties that will be used later.

3.1 How Human Beings Perceive a Scene

Figure 3.1 is a sketch of the flow chart of human visual processing. The visual process starts at the eyes where the lights enter, and ends at the brain after the visual information is passed on from the optic nerves. In this section, first we

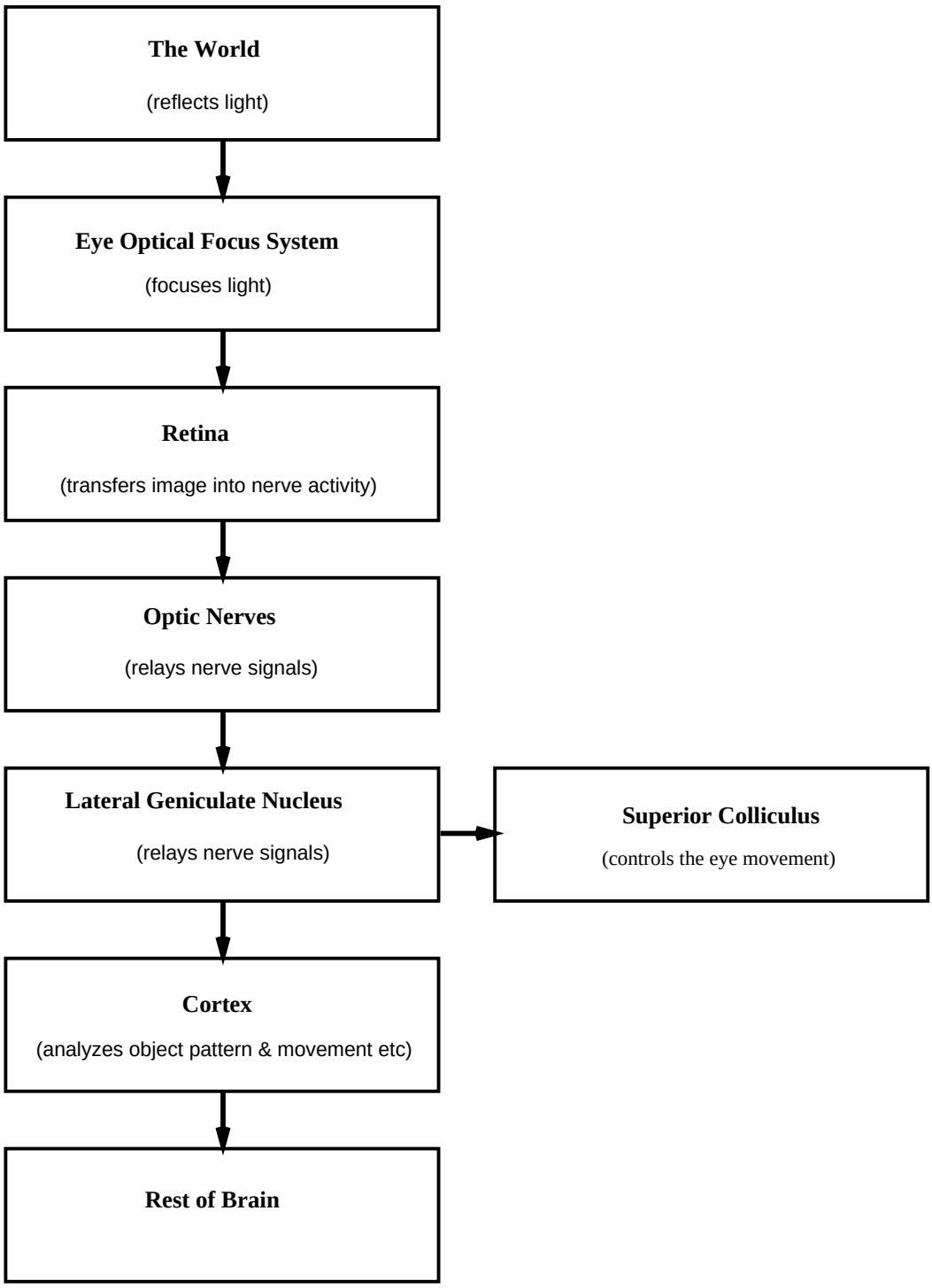


Figure 3.1: The Visual Process

explore the eye structure. Then, we discuss the visual processing in higher levels, such as in the cortex.

3.1.1 Eye

Figure 3.2 is a diagram of a horizontal cross section of the human eye. An eye has the following main components: cornea, aqueous humour, iris, pupil, lens, vitreous humour, retina, and optic nerves [JW75, pages 5-21].

The cornea, which covers the iris and pupil, is the transparent part of the coat of the eyeball. The tear ducts and eyelids keep it clean. The cornea scatters some portions of the incoming lights, refracts other portions of lights into the interior of the eye, and focuses the lights almost onto the retina. The remaining focus is provided by the lens. The aqueous humour is the fluid between the cornea and lens. If there is too much aqueous humour, the cornea is stressed outwards and the image is focused in front of the retina, while if there is too little aqueous humour, the cornea becomes flatter and the image is focused behind the retina.

The pupil is the small hole of the iris diaphragm through which light enters the lens. The iris diaphragm controls the diameter of the pupil based on the light intensity. The diameter becomes bigger when the light is dim and becomes smaller when the light is bright. When the pupil is small, only the centre of the lens, which is optically the best part of the lens, is used. Hence, the pupil regulates the light entering the eye, helping to produce a sharp image.

The lens is between the aqueous humour and the vitreous humour. The aqueous humour's pressure also adjusts the shape of the lens. The radial muscles tend to flatten the lens and the sphincter muscles tend to fatten it. These two muscles cooperate to adjust the focal length of the lens. As indicated in the

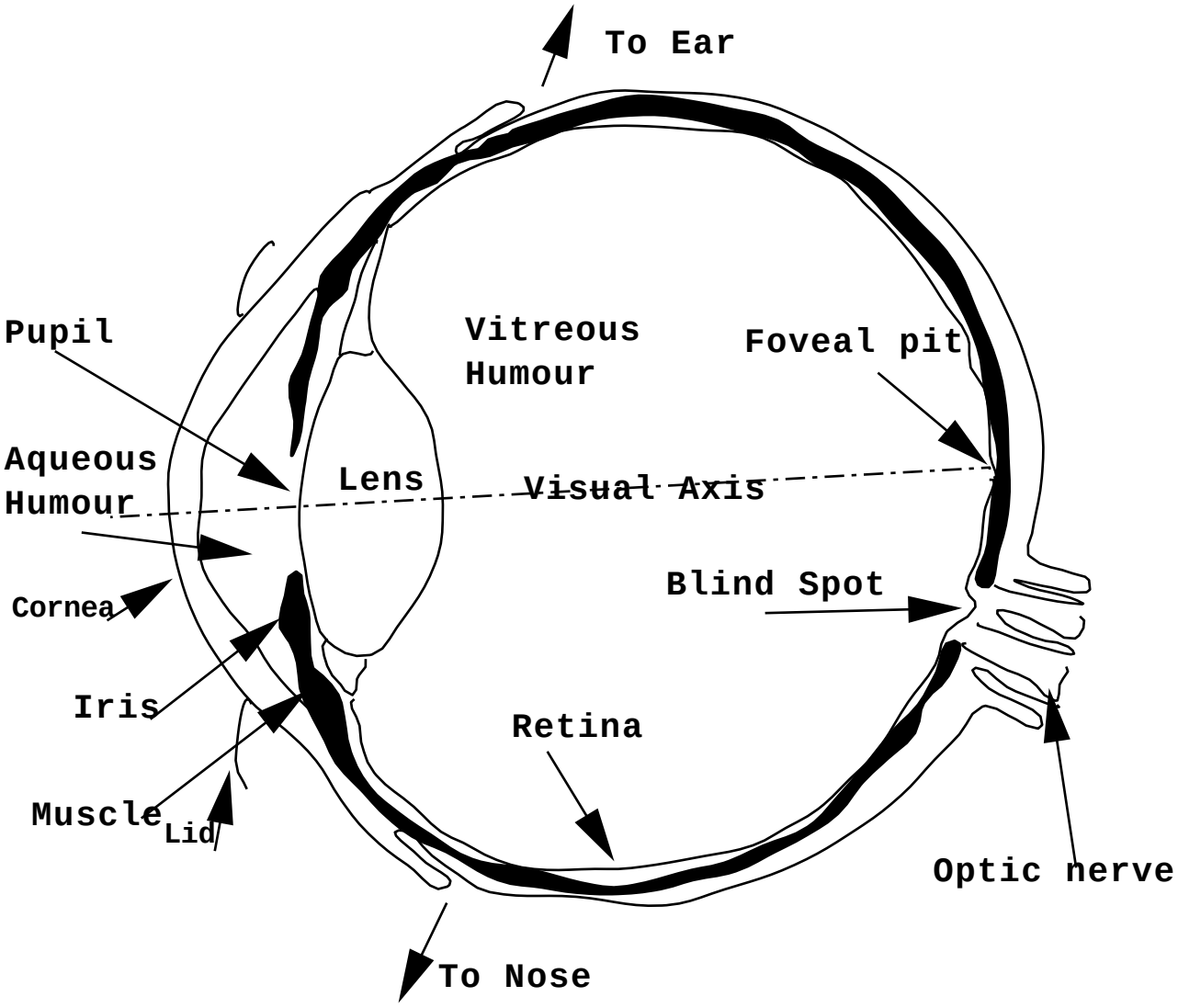


Figure 3.2: A Cross Section of The Eye [JW75, page 6][Cor70, page 40]

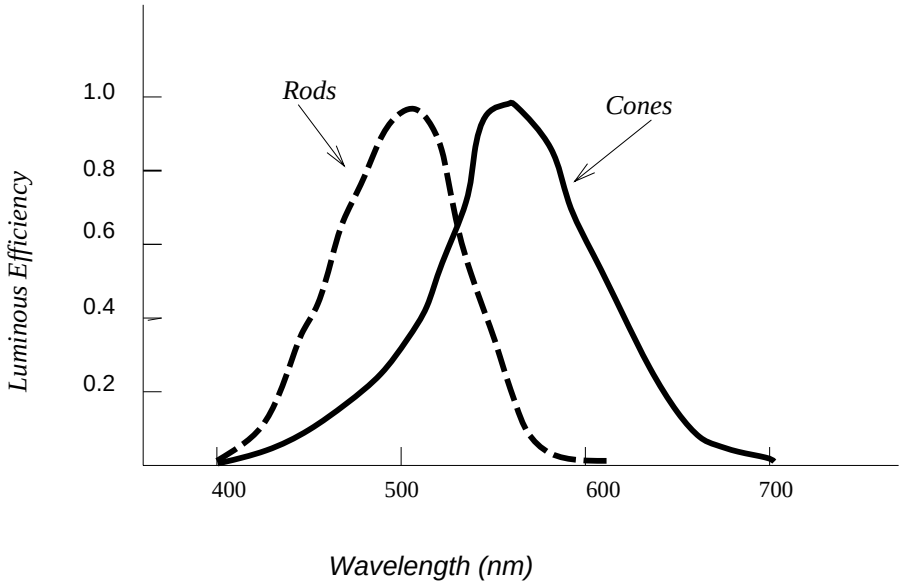


Figure 3.3: Spectral Sensitivities for Rods and Cones [JW75, page 9]

earlier paragraph, the cornea has already focused the image almost onto the retina, the lens with its flexible focal length adjusts the image precisely to the retina. However, both the shapes of optically effective surfaces in the eye and the distribution of refractive power within the lens are not optically perfect, which results in some amount of aberration.

The vitreous humour is a viscous fluid filling the interior of the eye. It maintains the distance from the lens to the retina close to a constant. The vitreous humour has large light-scattering tissues and fine light-scattering particles floating in it. They degrade the image quality on the retina, especially when the large light-scattering tissues are near to the retina. As we will point out later, this contributes to the fact that the eye is not a simple optical system.

The retina, the light-sensitive part, is at the back of the eyeball. Two types of cells (the rods and cones), retinal processing cells (including horizontal amacrine

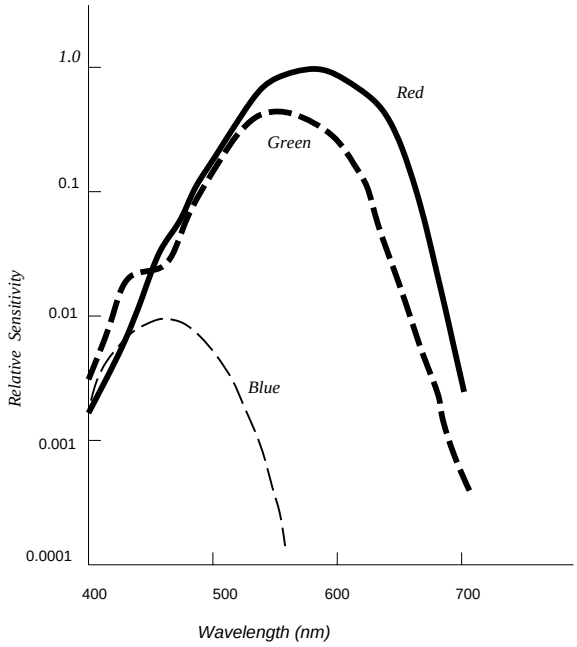


Figure 3.4: Spectral Sensitivities for Three Cones [CSW87, page 31]

and bipolar), the optic nerves, and the blood tissues spread over the retina. The rods and cones are non-uniformly distributed on the retina. At the centre of the retina, 1^o in diameter, called the foveal pit, the retina has the highest density of cones. There are no rods in the centre 2^o. The rods' density increases from 2^o and reaches a maximum at 20^o. Cones are found everywhere in the retina, but most densely within 5^o. Points close to the foveal pit, where the optic nerves leave for the brain, have no rods or cones. The region of these points is called the blind spot. If an object forms an image on the blind spot, it can not be seen.

Visual perception is indirectly determined by the outside world, but is directly determined by the retina image. If two different objects form the same image on the retina, human beings are not able to distinguish between these two objects. The cones and rods transfer the optical image into a pattern of nerve activity. Most

of the cones have direct optic nerves connected to the brain, while a bunches of rods share a nerve. The nerve activities are propagated from these cells to the brain through the optic nerves. When the light intensity is low, rods are more sensitive than cones. However, when the light intensity is high, the opposite is true. The rods and cones have different spectral sensitivities as shown in Figure 3.3. The rods are more sensitive than the cones to short wavelength light and less sensitive than the cones to long wavelength light. Unlike the rods, which can only yield black and white perception, the cones can produce chromatic perception. When the light is very dim, the rods are active instead of the cones, and one sees objects as gray instead of coloured. There exist three different types of cones. They are named the red, blue, and green cones because of the peaks of their spectral sensitivities. Figure 3.4 shows the spectral sensitivities of red, green, and blue cones.

The optical focus system in the eye, which is made up by the cornea and the lens etc., is not a perfect (or simple) geometrically optical system. It has diffraction, aberration, and scattering [Cor70, pages 56-60]. These imperfections differentiate it from a simple optical system and have significant influence on perception.

3.1.2 Higher Level Visual Processing

As stated in the previous subsection, the cones and rods transfer the image on the retina into patterns of optic nerve activities. The optic nerves pass these nerve activities to the lateral geniculate nuclei (LGN) [CSW87, page 67]. The LGN relays the signals from the optic nerves to the visual cortex, where higher level visual processing occurs, and to the superior colliculus, which controls eye

movements.

The cortex analyzes the object pattern and movement etc. It is located at Brodmann's areas 17, 18, and 19 [CSW87, pages 117-120]. Area 17 receives input from the LGN, and relays it to areas 18, 19, and the rest of the brain. It is organized as a retinotopic map, i.e. a cell in area 17 topologically corresponds to the part of the visual field where its receptive field is found. The left half of the visual field corresponds to the right half of the cortex. Areas 18 and 19 receive information both from area 17 and the LGN. Areas 18 and 19 consist of several retinotopic maps respectively. It is speculated that each retinotopic map extracts specific information from the input signals.

The physiology of higher level visual processing, such as visual cortex, is not very well understood. Visual processing can be viewed as a function which maps outside world to knowledge representation in the brain [Cow93]. If we regard a function as a black box and explore the properties of input and output for the function, we can acquire properties of that function. The psychophysical approach follows this paradigm. Some useful results will be addressed in the next sections.

3.2 The Interaction of Brightness and Spatial Relation

It is natural to believe that the brightness a viewer perceives is proportional to the intensity of a visual field. Unfortunately, it is not true. Many examples prove that perceived brightness depends on more than intensity alone.

In Figure 3.5, the three small squares have exactly the same level of intensity,

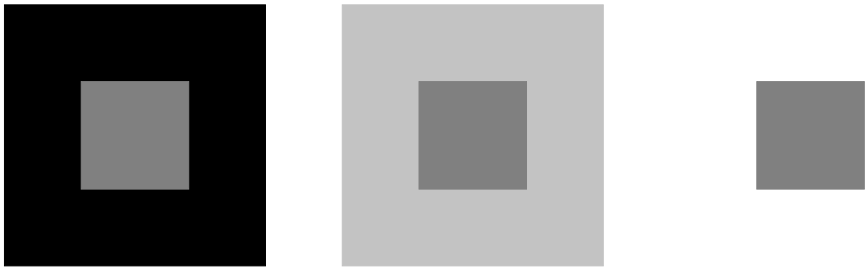


Figure 3.5: Brightness Is Not A Simple Function of Intensity [Cor70, page 279]

but we perceive that the square with black background is the brightest and the square with white background is the darkest.

Suppose two areas, separated by an edge, have the intensity distribution shown in Figure 3.6. Each area has a uniform intensity distribution. However, examining how perceived brightness varies, we see that it is not evenly distributed. The black area appears darker closer to the edge between the two areas. Similarly, the gray area appears brighter closer to the edge between the two areas. This effect was first described by Ernst Mach in 1865, and the pattern we see is called the Mach Band [Cor70, pages 276-277] [Rat72].

These two examples stated in previous paragraphs addressed an important perception concept, simultaneous contrast. The brightness of any object does not simply depend on the intensity of that object, but is affected by intensities of other objects close to this object. Heinemann's experiment [Cor70, page 279] gives further evidence that brightness is not a simple function of intensity.

The experiment was set up as in Figure 3.7. Heinemann fixed I_t and I_b, and then asked a subject to adjust I_m to match I_t. The results for $I_t = 100$ with various I_b's are illustrated in Figure 3.8. It shows that when I_b increases, starting from 0, I_m also increases and its value is bigger than 100. This does not follow

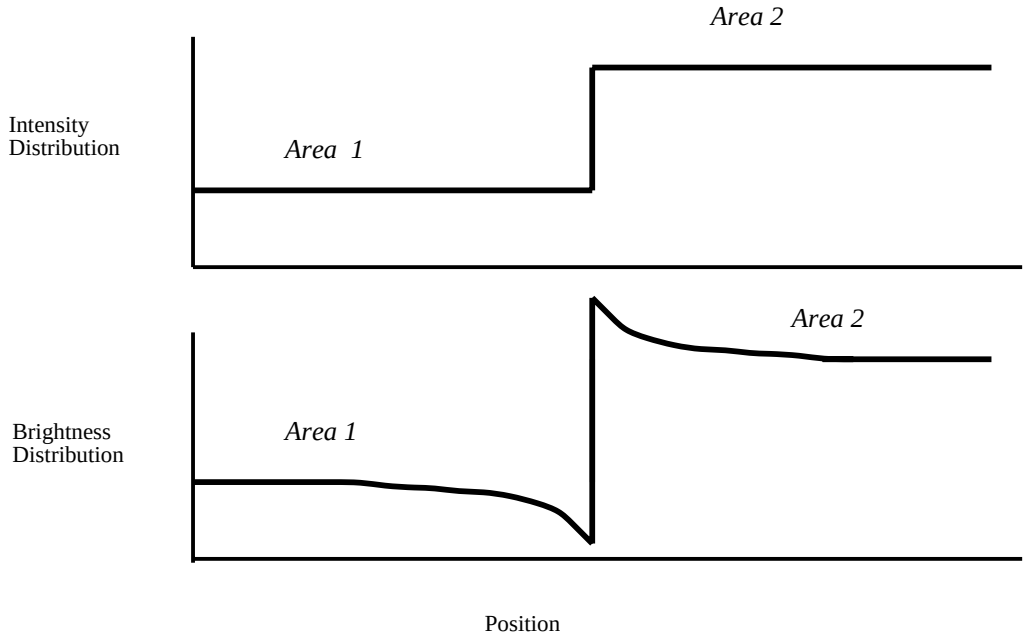


Figure 3.6: The Mach Band Effect

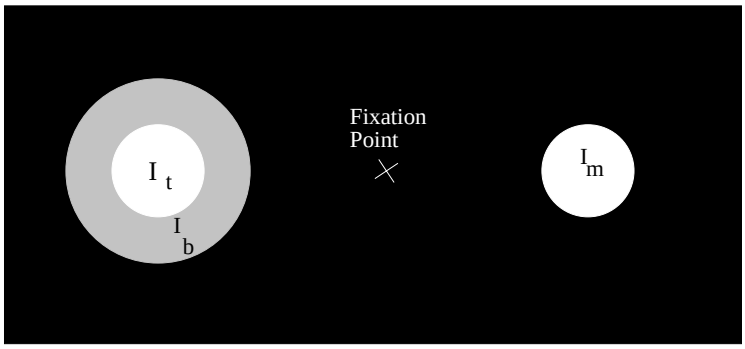


Figure 3.7: The Heinemann's Experiment [Cor70, page 279]

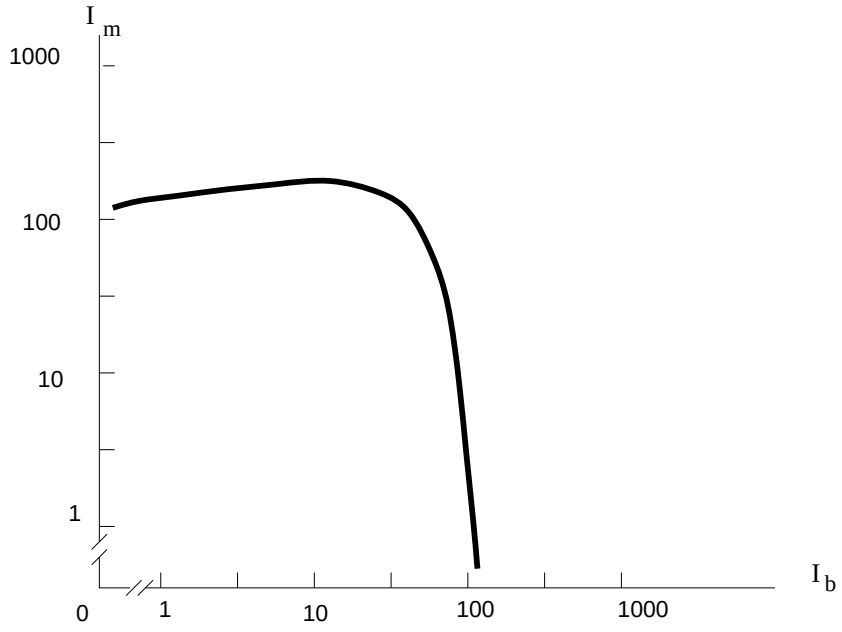


Figure 3.8: Results of Heinemann's Experiment ($I_t = 100$) [Cor70, page 280]

the "contrast" effect. However, as I_b reaches a certain value between 10 and 100, I_m starts to decrease. It becomes 0 when I_b is 110.

There are few physiological experiments on the human visual system that examine brightness perception. However, there are interesting experiments on limulus, a marine animal [Cor70, pages 285-310] [Rat72]. A limulus eye has many facets, each of which has its own retina. In some sense, a facet of a limulus is similar to a receptor on the retina of a human being. So, to some extent, the visual system of a limulus is similar to the visual system of a human being. Therefore, experiments on limulus might help to explain the brightness perception of human beings. This experiment measures the frequency of firing on an optic nerve connected to a facet when the facet is illuminated by a light. The frequency of firing decreases, when some light also illuminates a facet in

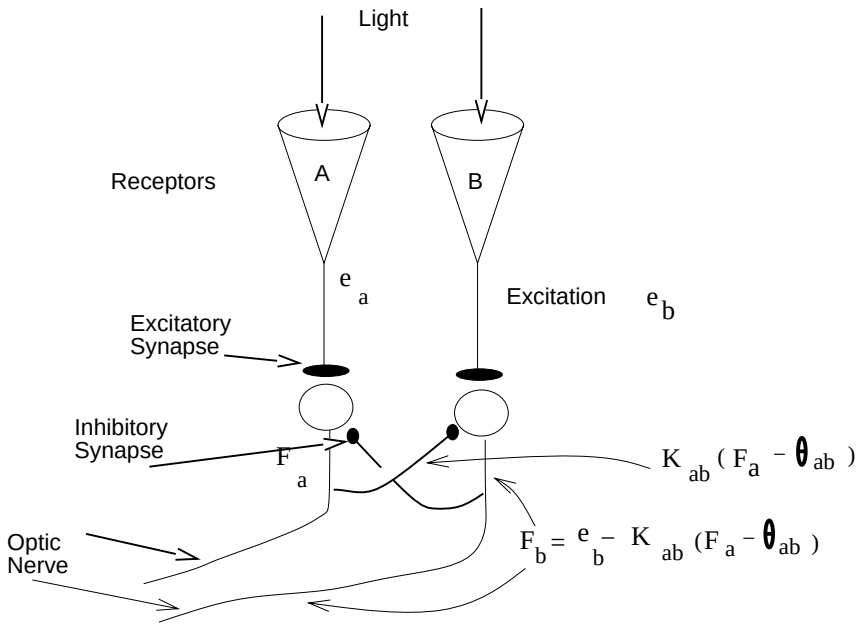


Figure 3.9: Inhibition Between Receptors [Cor70, page 295][Rat72]

the neighbourhood. To explain this phenomenon, a simple neural inhibition model was proposed as in Figure 3.9 [Cor70, page 295-304]. A light excites the receptor A. When the excitation exceeds a threshold, it is relayed through the **excitatory synapse** and the optic nerve fires in frequency F_a. If F_a is above another threshold θ_{ab}, the fires is relayed through a **inhibitory synapse** to the optic nerve of a neighbouring receptor B. As a result, the frequency of fires of B's optic nerve is reduced.

This experiment helps to explain brightness contrast. When the neighbouring area of area A has high intensity, the receptors for that area yield higher frequency of firing on their optic nerves, reducing the frequency of fires on the optic nerves of the receptors for area A. Therefore, A looks dimmer.

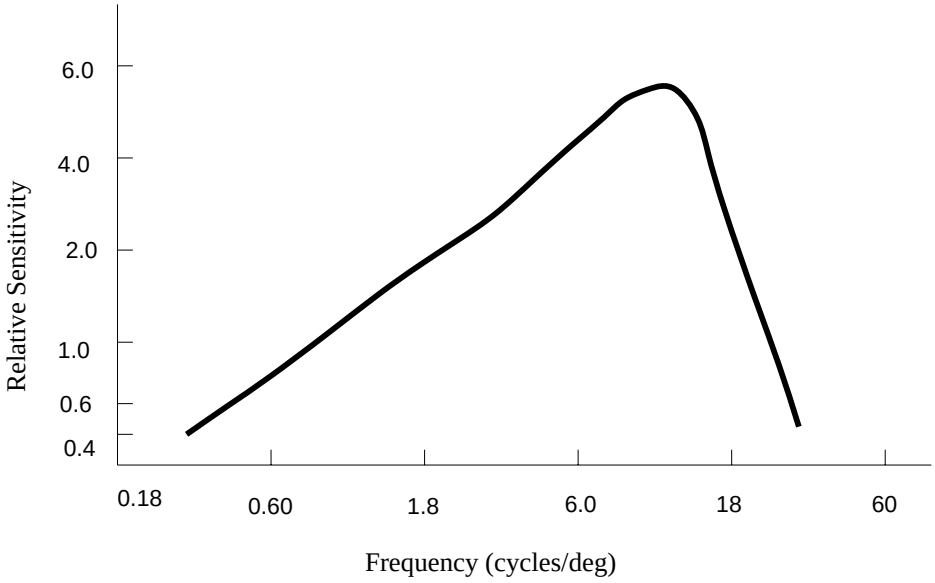


Figure 3.10: Modulation Transfer Function of A Human Eye [Cor70, page 341]

3.3 The Sensitivity of Human Vision with Intensity Variation

Using Fourier analysis, any pattern of light and dark can be represented as a sum of sinusoidal waves with various frequencies. A modulation transfer function specifies the sensitivity of a lens with respect to light waves of different frequencies [Cor70, pages 312-330].

The lens of a human eye also has a modulation transfer function. Figure 3.10 depicts such a function based on an experiment [Cor70, pages 330-342]. From this curve, we see that human vision is not sensitive to low frequency light waves, neither is it sensitive to very high frequency light waves. However, it is sensitive to high frequency light waves. Therefore, human vision is not sensitive to spatial intensity variation of low or very high frequency, but it is sensitive to spatial

intensity variation of high frequency.

These high frequencies stated in the previous paragraph define edges, and edges define shapes, the most important properties of objects. It is known that contrast helps to extract shape information. Thus, we conclude that retaining contrast at edges is usually important in visual perception.

Chapter 4

A Model of Constraint-Based Rendering

Chapter 2 reviewed rendering techniques and current research on rendering, and pointed out that there is little or no research on display contrast limitations, nor is there much or any research on rendering which uses the human vision properties. Chapter 3 discussed some properties of human vision system. This chapter proposes a model for constraint-based rendering, which tries to incorporate device contrast limitation into rendering by utilizing the properties of the human visual system.

4.1 Incorporating Device Limitation and Visual Perception into Rendering

As discussed in Chapter 2, a CRT cannot display absolute black, because the light reflects from the phosphor within the CRT and ambient light scatters over the

CRT's display surface. Because of of this restriction, the available contrast on a CRT is limited. The contrast between two points A and B can be calculated by the Michelson definition as in Equation 4.1.

$$C_{A,B} = \frac{L_A - L_B}{L_A + L_B} \tag{4.1}$$

where L_A and L_B are the luminances at A and B respectively and $L_A \leq L_B$ (when $L_B > L_A$, just switch L_A and L_B). If we could display absolute dark on a CRT, i.e. we would be able to achieve $L_{min} = 0$, then we could attain a $C_{A,min}$ of 1, where L_A is any luminance. Unfortunately, L_{min} can never be zero, and $L_{max} - L_{min} < L_{max} + L_{min}$. Therefore $C_{max,min}$ is always less than 1 on a CRT.

From Chapter 2, we know that the minimum luminance attainable on a CRT is from $\frac{1}{200}$ to $\frac{1}{40}$ of the maximum luminance attainable. From our experience, when there is ambient light in the room, the minimum luminance attainable on a CRT is usually higher than $\frac{1}{40}$, and is from 5% to 15% of the brightest luminance attainable. Suppose the darkest and brightest luminance attainable on a CRT is D_{max} and D_{min} (here D means Device), how can we render a scene whose ratio of maximum and minimum luminances are much bigger than D_{max}/D_{min}, such as depicted in Figure 4.1(a)?

There are two straightforward approaches. One is to replace all luminances greater than the maximum luminance by the maximum available luminance, and all the luminances below the minimum available luminance by the minimum luminance. This method is called clipping and is shown in Figure 4.1(b). It has a very obvious drawback that all the features whose luminances are out of the device luminance domain are cut off.

The other approach is to map the highest luminance in the image to the maximum attainable luminance on the CRT, and the lowest luminance in the image

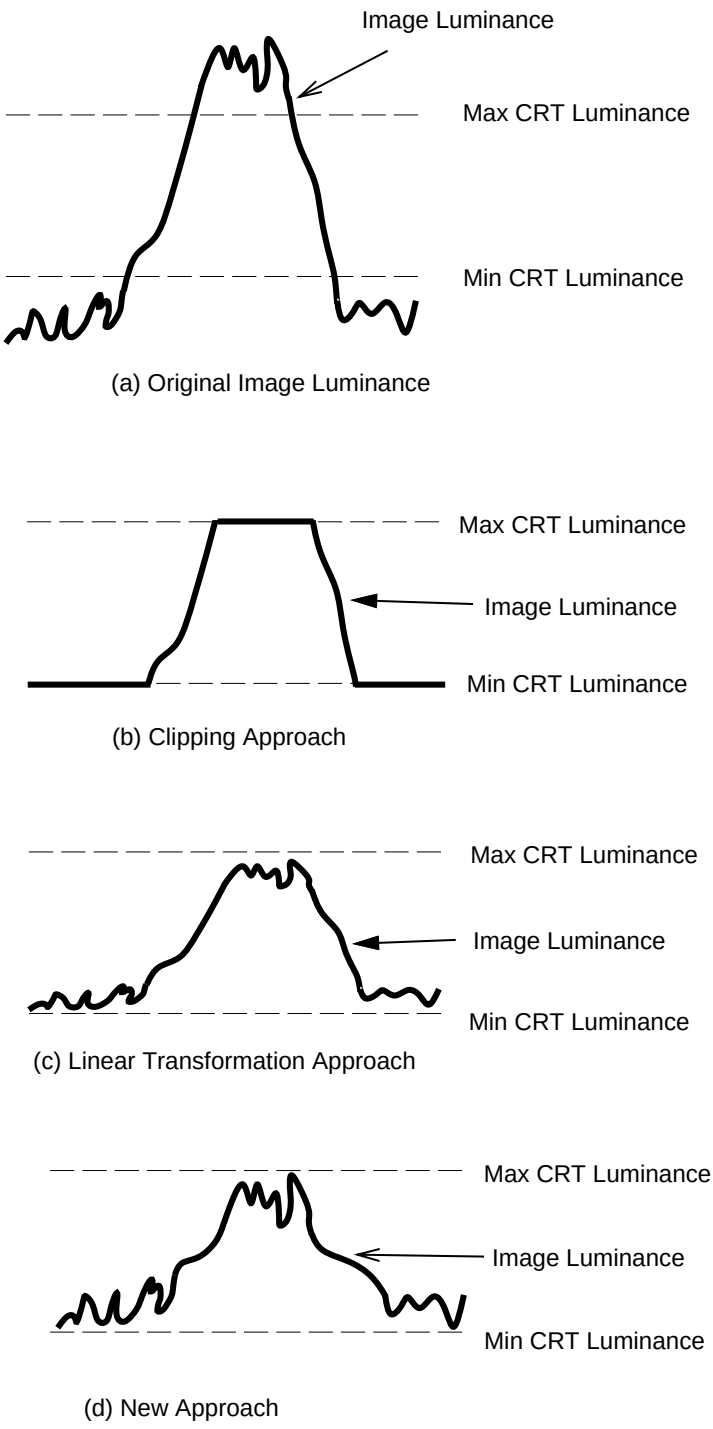


Figure 4.1: Dynamic Ranges of Image and Device

to the minimum attainable luminance on the CRT, and then linearly interpolate the other luminances (See Figure 4.1(c)). Suppose the highest and lowest luminances in the image are I_{max} and I_{min} (here I means Image) respectively, and the maximum and minimum attainable luminances on the CRT are D_{max} and D_{min} respectively, then any luminance I in the image and the corresponding device luminance D have the following relationship:

$$\begin{aligned} D &= D_{min} + \frac{D_{max} - D_{min}}{I_{max} - I_{min}}(I - I_{min}) \\ &= aI + b \end{aligned} \tag{4.2}$$

where $a = (D_{max} - D_{min})/(I_{max} - I_{min})$ and $b = D_{min} - aI_{min}$. If we have two points with image luminances I_1 and I_2 respectively, assuming $I_1 > I_2$, their contrast is $(I_1 - I_2)/(I_1 + I_2)$. We would like to know whether the contrast increases or decreases after the linear transformation, i.e. which is bigger between $(D_1 - D2)/(D_1 + D_2)$ and $(I_1 - I2)/(I_1 + I_2)$. First, let's state two propositions.

Proposition 4.1 *$D_{min}I_{max} - D_{max}I_{min} > 0$ when $I_{max}/I_{min} > D_{max}/D_{min}$, i.e. when the dynamic range of the image is higher than the dynamic range of the device.*

Proposition 4.2 *$b > 0$ when $I_{max}/I_{min} > D_{max}/D_{min}$ where b was defined in Equation 4.2.*

Proof.

$$\begin{aligned} b &= D_{min} - \frac{D_{max} - D_{min}}{I_{max} - I_{min}} I_{min} \\ &= \frac{D_{min}I_{max} - D_{max}I_{min}}{I_{max} - I_{min}} \\ &> 0 \qquad \{Proposition\ 4.1\} \end{aligned}$$

By applying Propositions 4.2, when $I_{max}/I_{min} > D_{max}/D_{min}$, we have:

$$\begin{aligned}
\frac{D_1 - D_2}{D_1 + D_2} &= \frac{aI_1 + b - aI_2 - b}{aI_1 + b + aI_2 + b} \\
&= \frac{a(I_1 - I_2)}{a(I_1 + I_2) + 2b} \\
&< \frac{a(I_1 - I_2)}{a(I_1 + I_2)} \qquad \{Proposition\ 4.2\} \\
&= \frac{I_1 - I_2}{I_1 + I_2}
\end{aligned} \tag{4.3}$$

We see that the contrast decreases after the linear transformation. We may replace $<$ by $\ll$ in the above inference. That means $(D_1 - D_2)/(D_1 + D_2) \ll (I_1 - I_2)/(I_1 + I_2)$ when $D_{max}/D_{min} \ll I_{max}/I_{min}$. Therefore, the contrast between two points is drastically reduced when the dynamic range of the image is much higher than that of the CRT. For example, comparing Figures 4.1(a) and 4.1(c), some features on the curve in Figure 4.1(a) are almost invisible in Figure 4.1(c).

Both of the two approaches discussed above result in the loss of information. From Chapter 3, we already know that the absolute value of luminance is not very important, but the contrast among the nearby luminances is important. We propose a new framework based on this concept. It considers the relationship among neighbouring objects. It relaxes the contrast where there is not much information, i.e. the luminance does not vary much, and keeps the contrast where the density of information is high, i.e. the luminance varies considerably (See Figure 4.1(d)).

In this new framework, we model the scene as a collection of polygons. At present, it only works for flat or Gouraud shaded scenes. Each polygon vertex has its original R, G, and B values as in conventional rendering. These values can be user specified or the results of preprocessing. We impose a hierarchy of constraints, with required constraints, strong constraints, medium constraints,

and weak constraints. The required constraint is that all the luminances are within the domain of the device contrast. A strong constraint is to maintain the contrasts across edges at the original value. A medium constraint is to keep the luminances in one polygon equal, and this constraint does not exist for Gouraud shaded scenes. A weak constraint is to preserve the luminance for a vertex at its initial value. In order to satisfy high priority constraints, we may have to sacrifice low priority constraints.

4.2 Measurement of Luminance and Contrast

This section will discuss the qualitative measurement of luminance and contrast to be used in the model described in the previous section.

4.2.1 Luminance

Although intensity, luminance, and brightness are closely related, they are in fact different in colour science. Intensity specifies the amount of electromagnetic energy in a light [BKT86, page 5-15]. The intensity of a light can be measured as

$$I = \int_{\lambda} \Phi_{\lambda} d_{\lambda} \tag{4.4}$$

where Φ_{λ} was defined in Chapter 2. The visual system is not equally sensitive to lights of all wavelengths [BKT86, page 5-15]. A luminous efficiency function specifies the effectiveness of different wavelengths to vision. $V'(\lambda)$ is the scotopic[1] relative luminous efficiency function, while $V(\lambda)$ is the photopic luminosity co-

[1] See [JW75, pages 352-353] for the definitions of scotopic and photopic visions.

efficient. These two functions are related to a standard scotopic/photopic observer's perception. Luminance is defined as

$$\begin{aligned} L &= \int_\lambda V'(\lambda)\Phi_\lambda d_\lambda \qquad or \\ L &= \int_\lambda V(\lambda)\Phi_\lambda d_\lambda \end{aligned} \tag{4.5}$$

for the scotopic or photopic system respectively. With a CRT, when the intensity of a pixel increases by a factor a, the number of photons in interval $[\lambda,\ \lambda + \Delta\lambda]$ approximately increases from Φ_λ to $a\Phi_\lambda$. Therefore, from Equations 4.4 and 4.5, we can see that $L \propto I$. Brightness is the attribute of a visual perception according to which a visual stimulus appears to be more or less intense, or according to which the the visual stimulus appears to emit more or less light [BKT86, page 9-3]. The brightness of a light source can be approximated by

$$B \approx L^{1/3}. \tag{4.6}$$

However, from Chapter 3, we know that brightness is not a simple function of luminance because it is also affected by the neighbouring luminances.

As indicated in Section 4.1, each vertex has a 3-tuple $(R,\ G,\ B)$ as input for our model. We need to calculate the luminance from this 3-tuple. There are variety of colour models ([FvDFH90, pages 579-600] and [SCB87]). Each has a 3-tuple to represent a colour and has one dimension to represent the luminance or has a formula to calculate the luminance from the 3-tuple. Table 4.1 lists some of the frequently used colour models and their luminance calculation. We choose the HSV (Hue, Saturation, and Value) colour model to measure the luminance for three reasons. The first is that the model has a single dimension V that varies with luminance. We only need one variable V instead of R, G, and B for a vertex to compute the luminance and contrast, which reduces our problem size by a

```
void HSV2RGB(h, s, v, *r, *g, *b) {
  if (s == 0) *r = *g = *b = v;  /* gray */
  else {
    if (h == 360) h = 0;        /* h is in [0, 360) */
    h /= 60;                    /* h is in [0, 6)   */
    i = Floor(h);
    f = h – i;
    p = v * (1 – s);
    q = v * (1 – s * f);                                          10
    t = v * (1 – s * (1 – f));
    switch (i) {
      case 0: (*r, *g, *b) = (v, t, p); break;
      case 1: (*r, *g, *b) = (q, v, p); break;
      case 2: (*r, *g, *b) = (p, v, t); break;
      case 3: (*r, *g, *b) = (p, q, v); break;
      case 4: (*r, *g, *b) = (t, p, v); break;
      case 5: (*r, *g, *b) = (v, p, q); break;
    } /* switch */
  } /* else */                                                     20
} /* HSV2RGB */
```

Figure 4.2: Transformation RGB to HSV [FvDFH90, page 592]

Name	Note	Luminance
CIE XYZ		Y
RGB	Red, Green, and Blue	some norm of (R, G, B)
CMY	Cyan, Magenta, and Yellow	some norm of (C, M, Y)
HSV	Hue, Saturation, and Value	V
HLS	Hue, Lightness, Saturation	L
YIQ		Y
CIE LAB		L

Table 4.1: Colour Models

factor of 3. The second reason is that the transformation from RGB to HSV and vice versa is computationally simple and efficient. The third reason, but not least, is that the HSV model is prominent in the computer graphics literature [SCB87].

The HSV colour model was first introduced by Smith [Smi78, SCB87, FvDFH90]. Hue corresponds to a position in the colour spectrum. Red, yellow, and violet etc. are some sample hue names. Saturation specifies the sharpness of the colour. A pure spectral colour is saturated, while gray is a desaturated colour. As indicated earlier, value approximates the luminance of the colour. Suppose $R, G, B \in [0, 1]$, the value V is the maximum of R, G, and B. Saturation and hue are the polar coordinates r and θ for the projection with respect to the point $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, when the point (R, G, B) is projected to the plane $R + G + B = 1$. Figures 4.2 and 4.3 gives the pseudo C code of transformation from RGB to HSV and vice versa ([FvDFH90, pages 592-593] and [SCB87]).

In actuality, the value V described here only approximates the drive luminance L_d [Mac91, MC92], which controls the digital to analog conversion (DAC).

```
void RGB2HSV(r, g, b, *h, *s, *v) {
  *v = Max(r, g, b);
  min = Min(r, g, b);
  delta = *v − min;
  if (*v == 0) *s = 0;     /* dark gray */
  else *s = delta / *v;
  if (*s == 0) *h = UNDEFINED;
  else if (r == *v)        /* between yellow and magenta */
      h = (g − b) / delta;
  else if (g == *v)        /* between cyan and yellow */
      h = 2 + (b − r) / delta;
  else                     /* between magenta and cyan */
      h = 4 + (r − g) / delta;
  h *= 60;                 /* convert to degree */
  h = (h >= 0) ? h : h + 360;   /* non−negative */
} /* RGB2HSV */
```

Figure 4.3: Transformation HSV to RGB[FvDFH90, page 593][SCB87]

The luminance we are concerned is the emitted luminance L_e [Mac91, MC92], and $L_e \propto V^\gamma$, where γ is the parameter of Gamma correction as discussed in Chapter 1. Because our primary interest is the contrast, we will see in next subsection that using V to approximate the luminance without considering γ suffices.

4.2.2 Contrast

There are quite a few definitions of contrast for two points A and B. For example, one definition of contrast is L_B/L_A assuming $L_A \leq L_B$ (if $L_A > L_B$, just switch L_A and L_B). In fact, all the contrast definitions are functions of L_B/L_A. For instance, in the Michelson definition of contrast, if we divide both the numerator and denominator by L_A, we have:

$$\frac{L_B - L_A}{L_B + L_A} = \frac{(L_B/L_A) - 1}{(L_B/L_A) + 1} = f(\frac{L_B}{L_A}) \tag{4.7}$$

Therefore, we use L_B/L_A as the definition of contrast. From now on, when we state contrast without explicitly mentioning the other definitions, we refer to this definition. In Subsection 4.2.1, we use V to approximate the drive luminance, and also have $L_e \propto V^\gamma$. Thus, following equation holds:

$$\frac{L_B}{L_A} \propto \left(\frac{V_B}{V_A}\right)^\gamma \tag{4.8}$$

From Equation 4.8, we can see that if we preserve the ratio of V_B/V_A, the contrast L_B/L_A is also maintained. Therefore, we use V_B/V_A to measure the contrast. Furthermore, we restrict contrast within the interval $[1, \infty)$, i.e. we always let the smaller value be the denominator and the larger value be the numerator.

4.3 The Rendering Pipeline

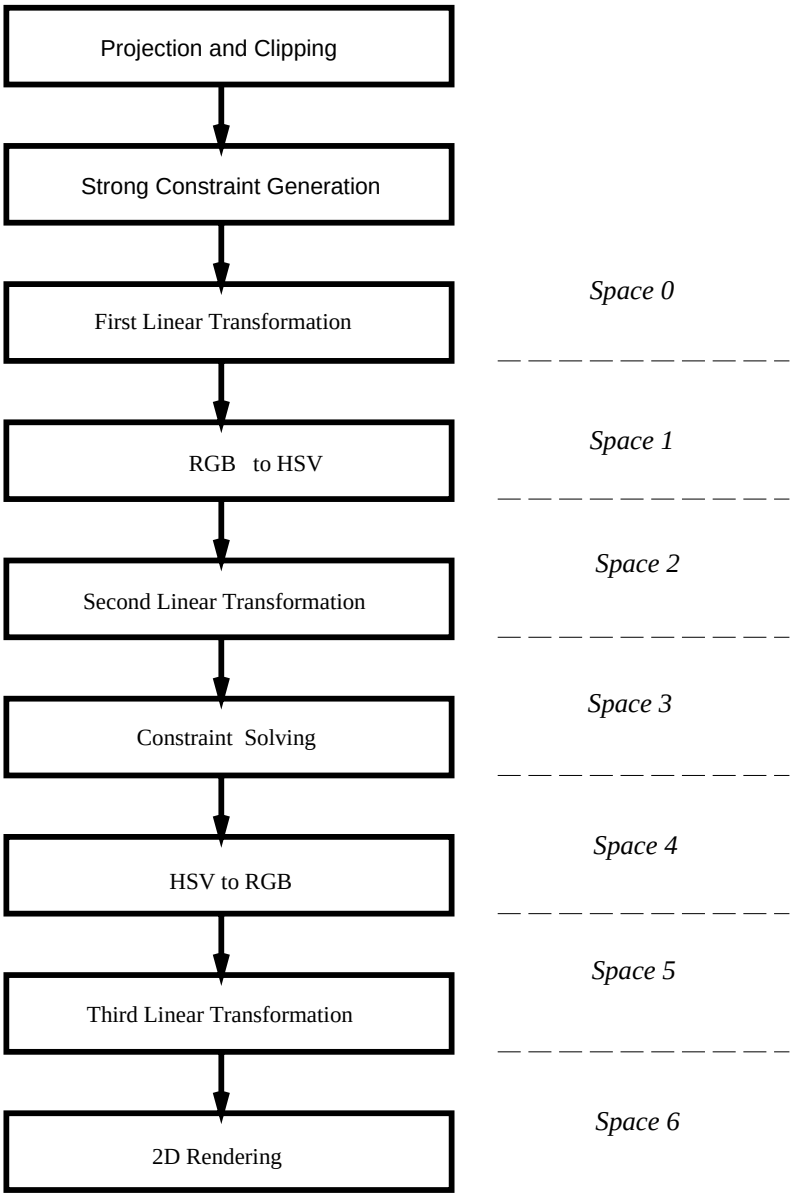


Figure 4.4: The Rendering Pipeline

It is the time to describe our rendering pipeline now that the relative concepts and measurement have been introduced. The rendering pipeline consists of following stages: projection and clipping, strong constraint generation, first linear transformation, RGB to HSV, second linear transformation, constraint solving, HSV to RGB, third linear transformation, and $2D$ rendering. Figure 4.4 illustrates this pipeline. For the convenience of further reference, we call the colour space before the first linear transformation as **Space 0**, the colour space after the first linear transformation but before the RGB to HSV transformation as **Space 1**, and so on as shown in the figure. We will discuss each step briefly in this section. Implementation details will be explained in the next chapter.

Projection and clipping project the $3D$ scene onto a $2D$ plane and clip the scene against a viewing volume in the homogeneous coordinate system[2]. $2D$ rendering fills the $2D$ polygons with the colours interpolated from the polygons' vertex colours. The $2D$ polygons are reconstructed through visible surface processing after the scene is projected onto $2D$.

Constraint solving not only solves the constraints, but also first adds the required, the medium, and the weak constraints to form the constraint hierarchy, together with the strong constraints created in strong constraint generation.

As we will discuss further in the next section and chapter, strong constraint generation generates the strong constraints from the relationship of all the edges in $2D$. The contrast to be maintained by a strong constraint is computed using the colour in Space 0 (the luminance is the maximum of the RGB 3-tuple). The strong constraints are generated through a modified hidden surface removing algorithm. Hence, a by product of generating the strong constraints is hidden

[2] Homogeneous coordinate system will be explained in Chapter 5.

surface removal.

The first linear transformation maps the input RGB into $[0,\ 1] \times [0,\ 1] \times [0,\ 1]$. Suppose the maximum and minimum of all the input R, G, and B's are $MaxRGB$ and $MinRGB$. The new R, G, and B are:

$$New\ R = \frac{R - MinRGB}{MaxRGB - MinRGB} \tag{4.9}$$

$$New\ G = \frac{G - MinRGB}{MaxRGB - MinRGB} \tag{4.10}$$

$$New\ B = \frac{B - MinRGB}{MaxRGB - MinRGB} \tag{4.11}$$

The second linear transformation transforms the luminance V of the HSV colour into the device-attainable dynamic ranges. If the minimal luminance attainable is $100 \times LowerBound\%$ of the maximal luminance available on a CRT, the new V is computed by:

$$New\ V = LowerBound + (1 - LowerBound)V \tag{4.12}$$

The third linear transformation is used to translate the RGB in $[0,\ 1] \times [0,\ 1] \times [0,\ 1]$ to $[0,\ MaxDeviceRGB] \times [0,\ MaxDeviceRGB] \times [0,\ MaxDeviceRGB]$, where $MaxDeviceRGB$ is the maximum of the values stored in the frame buffer to drive the digital to analog converter. For a CRT with 24 bitplanes, there are 8 bitplanes for **Red**, **Green**, and **Blue** respectively, and its $MaxDeviceRGB$ is 255. The device R, G, and B are:

$$Device\ R = Round(\ MaxDeviceRGB \times R\ ) \tag{4.13}$$

$$Device\ G = Round(\ MaxDeviceRGB \times G\ ) \tag{4.14}$$

$$Device\ B = Round(\ MaxDeviceRGB \times B\ ) \tag{4.15}$$

It should be noted that the first and second linear transformations affect the contrast between two points. This is the reason why the strong constraints are

necessary. On the other hand, the third linear transformation does not affect the contrast between two points as we can see in Equation 4.16.

$$\frac{MaxDeviceRGB \times V_B}{MaxDeviceRGB \times V_A} = \frac{V_B}{V_A} \tag{4.16}$$

4.4 Formalization of Constraints

As discussed earlier, a constraint hierarchy has four levels of constraints: the required constraints, the strong constraints, the medium constraints, and the weak constraints.

The required constraint is that the luminance must be within the device contrast limitation. For each vertex i in every polygon, this required constraint can be formalized as:

$$LowerBound \leq V_i \leq 1.0 \tag{4.17}$$

where $LowerBound$ is the minimum luminance available when the maximum luminance attainable is 1.0, and V_i is in Space 3.

The strong constraints maintain the contrast across edges at the contrast in Space 0. Edges only occur at polygon boundaries. Contrast along the edges helps a viewer distinguish different polygons, and thus enhances the presentation of information. We know that the luminance for a point A on a line segment P_1P_2 is $V_A = (1-\alpha)V_1 + \alpha V_2$ using flat and Gouraud shading models, where V_1 and V_2 are the luminances for P_1 and P_2 respectively, and $\alpha = |P_1A|/|P_1P_2|$ (See Figure 4.5(a)). As shown in Figure 4.5(b), the contrast for two points A and B across an edge formed by P_1P_2 and P_3P_4 is:

$$\frac{V_A}{V_B} = \frac{(1-\alpha)V_1 + \alpha V_2}{(1-\beta)V_3 + \beta V_3} = C_{A,B_0} \tag{4.18}$$

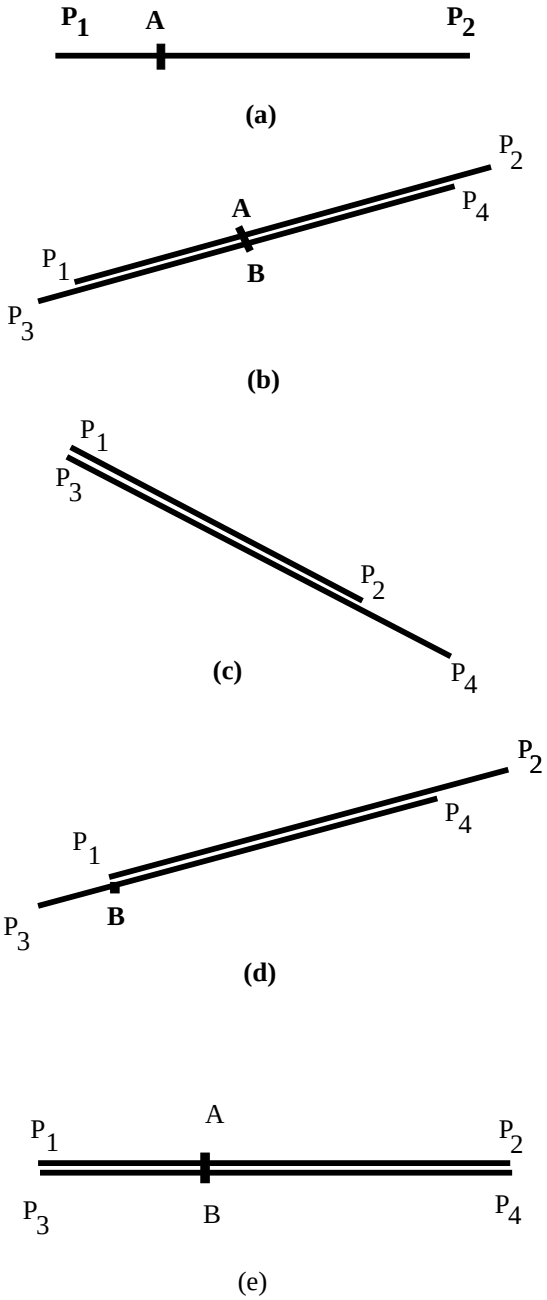


Figure 4.5: Strong Constraints

where $\beta = |P_3B|/|P_3P_4|$ and C_{A,B_0} is the original contrast between A and B in Space 0. In Figure 4.5, for illustration purposes P_1P_2 and P_3P_4 do not overlap, but in actuality they are co-linear and overlap. Equation 4.18 should be true for any $\alpha \in [0,\ 1]$. If we set α to zero, we get:

$$\frac{V_1}{(1-\beta)V_3 \ + \ \beta V_4} \ = \ C_{1,B_0} \tag{4.19}$$

where C_{1,B_0} is the contrast between P_1 and B in Space 0. Equation 4.19 can be split into equivalent two cases. The first case is that the projection of P_1 onto P_3P_4 is at P_3 or P_4. Without loss of generality, we assume the projection is at P_3. The β value is then equal to 0, and Equation 4.19 becomes (Figure 4.5(c)):

$$\frac{V_1}{V_3} \ = \ C_{1,3_0} \tag{4.20}$$

This is equivalent to:

$$V_1 \ - \ C_{1,3_0}V_3 \ = \ 0 \tag{4.21}$$

The other case is that the projection of P_1 onto P_3P_4 is between P_3 and P_4, i.e. β in Equation 4.19 is not 0 or 1 as in Figure 4.5(d). This can be transformed into the linear equation:

$$V_1 \ - \ C_{1,B_0}(1-\beta)V_3 \ - \ C_{1,B_0}\beta V_4 \ = \ 0 \tag{4.22}$$

If $V_{A_0} \ < \ V_{B_0}$ in Equation 4.18 where $V_{A_0} \ = \ Max(R_A, G_A, B_A)$ in Space 0 and similarly for V_{B_0}, the Equation 4.18 should be revised to:

$$\frac{V_B}{V_A} \ = \ \frac{(1-\beta)V_3 \ + \ \beta V_3}{(1-\alpha)V_1 \ + \ \alpha V_2} \ = \ C_{B,A_0} \tag{4.23}$$

and as a consequence, Equation 4.22 becomes:

$$-C_{B,1_0}V_1 \ + \ (1 \ - \ \beta)V_3 \ + \ \beta V_4 \ = \ 0 \tag{4.24}$$

We did not revise Equations 4.20 or 4.21 because it will not lead to a different format of equation. In fact, the first case is a special case of the second one, but it has the merit of only involving two variables.

We have shown that to maintain the contrast across an edge, either Equation 4.21, Equation 4.22, or Equation 4.24 must be satisfied. Now we will show that if the aforementioned equations are satisfied, the contrasts for mid-points are roughly maintained to their values in Space 0. When the shading model is flat shading, we have (at least we try to have by enforcing the medium constraints) $V_1 = V_2$ and $V_3 = V_4$, and it is easy to show that $V_A/V_B = V_{A_0}/V_{B_0}$. When the shading model is Gouraud shading, if the average luminance of A's polygon is U_a and the average luminance of B's polygon is U_b, we can write V_A as $U_a + \Delta V_A$, V_B as $U_b + \Delta V_B$, and so on. In a polygon with average luminance U, we may assume $|\Delta V_i|/U \ll 1$ for each vertex in that polygon, otherwise, we can subdivide the polygon until the assumption holds. Suppose $U_b < U_a$, we have:

$$\frac{V_1}{V_3} = \frac{U_a + \Delta V_1}{U_b + \Delta V_3} \approx \frac{U_a}{U_b} \tag{4.25}$$

$$\frac{V_2}{V_4} = \frac{U_a + \Delta V_2}{U_b + \Delta V_4} \approx \frac{U_a}{U_b} \tag{4.26}$$

Hence,

$$\frac{V_1}{V_3} \approx \frac{V_2}{V_4} \tag{4.27}$$

It is equivalent to[3]:

$$\frac{(1-s)V_1}{(1-s)V_3} \approx \frac{sV_2}{sV_4} \tag{4.28}$$

It is again equivalent to:

$$\frac{(1-s)V_1}{sV_2} \approx \frac{(1-s)V_3}{sV_4} \tag{4.29}$$

[3]In the following inference, $s \neq 0$ and $s \neq 1$. When $s = 0$, it is the simple case that $V_A = V_1$ and $V_B = V_3$, and when $s = 1$, $V_A = V_2$ and $V_B = V_4$.

Then, we have:

$$\frac{(1-s)V_1 + sV_2}{sV_2} \approx \frac{(1-s)V_3 + sV_4}{sV_4} \tag{4.30}$$

That is:

$$\frac{(1-s)V_1 + sV_2}{(1-s)V_3 + sV_4} \approx \frac{sV_2}{sV_4} = \frac{V_2}{V_4} \tag{4.31}$$

Follow the same reasoning, we can derive:

$$\frac{(1-s)V_{1_0} + sV_{2_0}}{(1-s)V_{3_0} + sV_{4_0}} \approx \frac{V_{2_0}}{V_{4_0}} \tag{4.32}$$

Therefore, in our problem domain, V_A/V_B is a reasonable approximation to V_{A_0}/V_{B_0} when $V_1/V_3 = V_{1_0}/V_{3_0}$ and $V_2/V_4 = V_{2_0}/V_{4_0}$.

For flat shading model, a medium constraint tries to keep all vertices in one polygon the same luminance. For any pair of vertices in each polygon:

$$V_i - V_j = 0, \qquad i \neq j \tag{4.33}$$

Unfortunately, this yields n^2 medium constraints for a polygon with n vertices. Instead we only impose the constraint that each vertex has the same luminance as its neighbouring vertices. For each vertex in every polygon:

$$V_i - V_{(i+1)} = 0, \tag{4.34}$$

where $i + 1 = 0$ when $i = n - 1$. This not only reduces the number of medium constraints from n^2 to n, but also has justification in visual perception. The visual system is sensitive to local variations, but not sensitive to slow variation across large area.

The weak constraints preserve the luminance for vertices close to their initial values in Space 3. Every vertex in each polygon has a weak constraint:

$$V_i = V_{i0} \tag{4.35}$$

where V_{i0} is the initial luminance for vertex i in Space 3.

It is desired to get a solution $\mathbf{V}$, where $\mathbf{V} = [V_0, V_1, ..., V_{num}]$ and num is the total number of vertices in the scene, within the interval defined by Equation 4.17 to satisfy all the linear equations derived from Equations 4.21, 4.22, 4.24, 4.34, and 4.35. Unfortunately, most of the time, this system of linear equations does not have a solution. Then a vector $\mathbf{V}$, which results in the minimum Euclidian norm of the residuals for all the equations, is the optimal choice. To achieve the minimum Euclidian norm is equivalent to achieve the minimum summation of squares of the residuals for all the equations. However, we wish to satisfy the equations derived from strong constraints more than other equations. This can be accommodated by first multiplying the equations of strong constraints by a large weight, the equations of medium constraints by a medium weight, and the the equations of weak constraints by a small weight, then finding the vector $\mathbf{V}$ resulting in minimum summation of squares of the residuals. In fact, this is called a bounded linear least squares (BLLS) problem and is well studied in scientific computation [GMSW84, GHMS86, Sto71]. Moreover, there is a commercial software package available to solve a BLLS problem [NAG91].

Chapter 5

Implementation

A prototype system of constraint-based rendering as addressed in Chapter 4 has been implemented on DECstation 5000/200 running ULTRIX and X Window System [Nye88, Nye90]. The prototype system is written in ANSI C [KR88] with the exception of the interface to the NAG package [NAG91], which is written in Fortran 77.

This chapter describes the implementation of this prototype system. It consists of seven functional blocks. They are (1) preprocessing, (2) input, (3) projection, (4) clipping, (5) constraint generation, (6) constraint solving, and (7) 2D rendering. The system architecture is shown in Figure 5.1. Note that the system architecture does not exactly follow the rendering pipeline stated in Chapter 4 because we compact some steps of the pipeline into one block for the convenience of implementation. Nevertheless, each of the steps in the pipeline addressed there are performed in the same order.

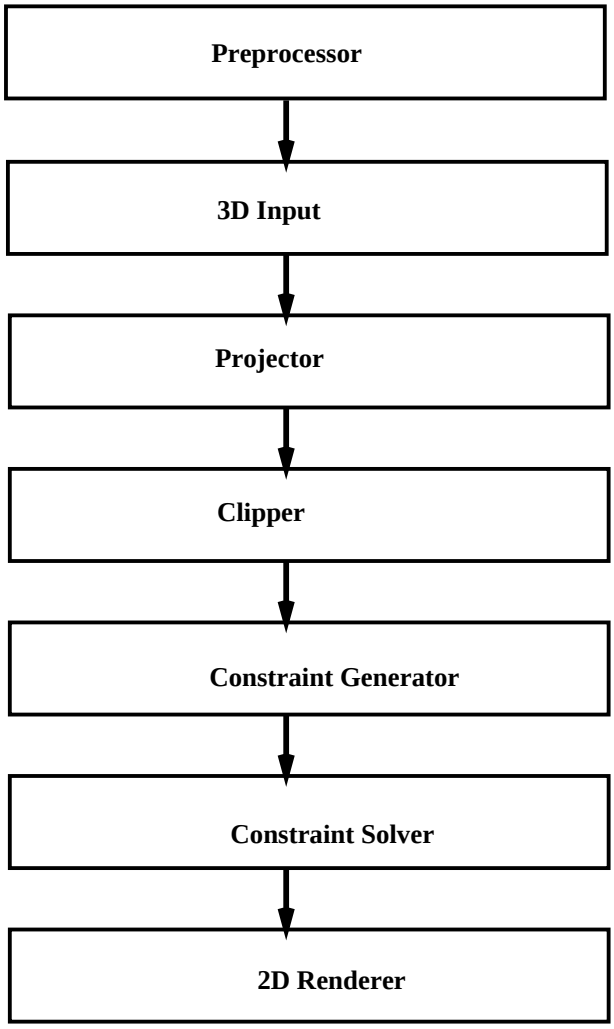


Figure 5.1: System Architecture of Constraint-Based Renderer

5.1 Preprocessing

The preprocessor reads in a scene description, and creates the input for the constraint-based renderer (CBR). The scene description consists of global ambient and diffuse lighting information, viewing parameters, and a list of objects. Each object has a list of polygons and the local lighting information for this object.

Viewing information is passed transparently while the polygonal objects are processed as follows: the R, G, and B values for all the vertices in each object are computed according to the shading model specified by a command line argument. The shading model is either flat shading or Gouraud shading. The preprocessor outputs a list of polygons. Each polygon is specified as a list of vertices. Every vertex has six real numbers as its attributes. The first three real numbers represent the X, Y, and Z coordinates for a vertex, while the last three specify the R, G, and B values for that vertex.

5.2 Input, Projection, and Clipping

Input, projection, and clipping are performed in the traditional approach. Background materials on projection and clipping can be found in [FvDFH90, pages 229-281] and [BN78].

5.2.1 Input

This part is very similar to the parser in the preprocessor. It reads in the input and stores information in internal data structures. The input consists of the viewing parameters such as the eye location, the viewing vector, the up vector etc. and

a list of polygons. Each polygon is represented by a list of vertices, and every vertex is specified by six real numbers as indicated in the output format of the preprocessor.

5.2.2 Projection

Before projection, back-face culling ([FvDFH90, pages 663-664]) is performed. A polygon is regarded as a back-face if its normal points away from the eye point. Mathematically, a polygon is back-facing if the dot product between its normal and the sight vector is positive. The normal for a polygon is computed by the following formulae:

$$Normal = \frac{1}{2}\sum_{i=0}^{n-1}(V_i \times V_{i+1}) \tag{5.1}$$

where $i + 1$ is in fact $(i + 1)\ mod\ n$, n is the number of vertices in the polygon, and V_i's are the vertices. Since we are only interested in the orientation of the normal, the constant $\frac{1}{2}$ is not necessary.

The input viewing parameters are: eye point (e_x, e_y, e_z), sight vector (s_x, s_y, s_z), up vector (u_x, u_y, u_z), the distance from near plane to the eye point n, the distance from far plane to the eye point f, the horizontal viewing angle α which decides the world window size from n, and the ratio r between X and Y in the world window.

A homogeneous coordinate system uses a 4 component vector to represent a point [FvDFH90, page 204, pages 213-214]. (x, y, z, w) is equivalent to $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$ in a Euclidian coordinate system. Using homogeneous coordinates, a geometrical transformation can be conveniently expressed as a 4×4 matrix.

$$(x, y, z, w) = (x', y', z', w')T \tag{5.2}$$

where T is a transformation matrix, $(x',\ y',\ z',\ w')$ is the coordinate before the transformation, and $(x,\ y,\ z,\ w)$ is the coordinate after the transformation. Appendix A lists a few matrices for a variety of modelling transformations and viewing transformations used in this implementation.

There are two basic classes of projections, perspective projection and parallel projection [FvDFH90, pages 229-271]. We use perspective projection in the implementation. When the eye point is at $(0,\ 0,\ 0)$, the sight vector is in the $-Z$ direction, and the up vector is at the Y direction, the projection is in the canonical form. Equation 5.3 is the projection matrix in the canonical form. Its derivation is given in Appendix A.

$$P \quad = \quad \begin{bmatrix} \frac{\cot(\frac{\alpha}{2})}{n} & 0 & 0 & 0 \\ 0 & \frac{r\cot(\frac{\alpha}{2})}{n} & 0 & 0 \\ 0 & 0 & \frac{f}{n(n-f)} & -\frac{1}{n} \\ 0 & 0 & \frac{f}{n-f} & 0 \end{bmatrix} \tag{5.3}$$

The eye point, the projection plane, and sight vector etc. are at arbitrary locations or in arbitrary orientation in $3D$. We first transform this into a canonical form by translating the eye point to $(0,\ 0,\ 0)$, rotating the sight vector into the $-Z$ direction, and rotating the up vector into the Y direction. This will yield a viewing transformation matrix $View$ as defined in Appendix A. After these viewing transformation steps, the vertices will be projected onto the near plane.

The projection procedure multiplies every vertex by the projection matrix to produce the projected vertices as in Equation 5.4.

$$(x'',\ y'',\ z'',\ w'') \ = \ (x,\ y,\ z,\ 1)\,(View\ P) \tag{5.4}$$

Clipping Bound	W.E.C.
Left	$w + x$
Right	$w - x$
Bottom	$w + y$
Up	$w - y$
Near	z
Far	$w - z$

Table 5.1: Window Edge Coordinates

5.2.3 Clipping

Clipping is the procedure of eliminating the object parts which lie out of the viewing volume. After the polygons pass through the projection stage, the polygons are represented by homogeneous coordinates [FvDFH90, page 204, pages 213-214]. The clipping can be done in homogeneous coordinate space effectively ([BN78] and [FvDFH90, pages 275-278]).

The viewing volume is defined as $[-1,\ 1] \times [-1,\ 1] \times [0,\ 1]$ in the projected space, i.e. a vertex $(x,\ y,\ z,\ w)$ is in the viewing volume if and only if it satisfies:

$$\begin{aligned} -1 &\leq \frac{x}{w} \leq 1 \\ -1 &\leq \frac{y}{w} \leq 1 \\ 0 &\leq \frac{z}{w} \leq 1 \end{aligned} \tag{5.5}$$

From Equation 5.3, it is easy to show that $w > 0$, so the above equations are equivalent to:

$$w + x \geq 0$$

$$\begin{aligned} w - x &\geq 0 \\ w + y &\geq 0 \\ w - y &\geq 0 \\ z &\geq 0 \\ w - z &\geq 0 \end{aligned} \tag{5.6}$$

Therefore, to see whether a vertex is to the right of the viewing volume's left bound, we only need to see if $w + x \geq 0$. $w + x$ is called the window edge coordinate (WEC) for the left bound. The window edge coordinates for other bounds are listed in Table 5.1. A vertex is in the viewing volume if and only if its window edge coordinates for all the clipping bounds are non-negative.

Now let us describe a polygon clipping algorithm. It is a $3D$ extension to the Sutherland-Hodgman polygon clipping algorithm discussed in [FvDFH90, pages 124-127]. Each of the six clipping bounds clips a polygon consecutively. The previous clipper's output is the next clipper's input. A clipper transforms an input vertex sequence into an output vertex sequence. At a vertex U in the input sequence, a clipper first computes the window edge coordinates for vertex U and vertex V which follows[1] U in the input sequence. If both WEC's are less than zero, the edge UV is out of the viewing volume, and the clipper outputs nothing. On the other hand, if both WEC's are non-negative, the entire edge is in the viewing volume. Hence the clipper outputs vertex V. If U's WEC is less than 0, but V's WEC is non-negative, the clipper outputs two vertices, the intersection between the clipping bound and edge UV and the vertex V. If U's WEC is non-negative, but V's WEC is less than zero, the clipper outputs the intersection between the

[1]The last vertex of the input sequence is followed by the first vertex of the input sequence.

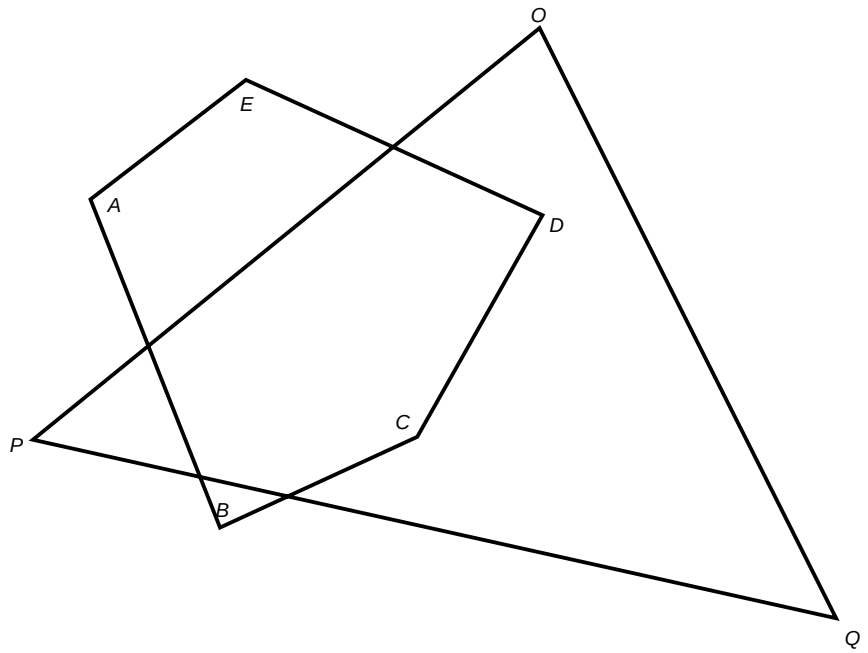


Figure 5.2: Polygons Overlap After Projection

clipping bound and edge UV. The intersection is calculated by Equation 5.7.

$$\begin{aligned} \rho &= \frac{WEC_u}{WEC_u - WEC_v} \\ x_i &= x_u + \rho(x_v - x_u) \\ y_i &= y_u + \rho(y_v - y_u) \\ z_i &= z_u + \rho(z_v - z_u) \\ w_i &= w_u + \rho(w_v - w_u) \end{aligned} \tag{5.7}$$

where $U = (x_u,\, y_u,\, z_u,\, w_u)$, $V = (x_v,\, y_v,\, z_v,\, w_v)$, the intersection is $(x_i,\, y_i,\, z_i,\, w_i)$, and WEC_u and WEC_v are the window edge coordinates for U and V respectively.

After the clipping is finished, the homogeneous coordinates are no longer needed. We translate the homogeneous coordinates back to the Euclidian coordinates. That is $(x'',\, y'',\, z'', w'')$ is replaced by $(\frac{x''}{w''},\, \frac{y''}{w''},\, \frac{z''}{w''})$.

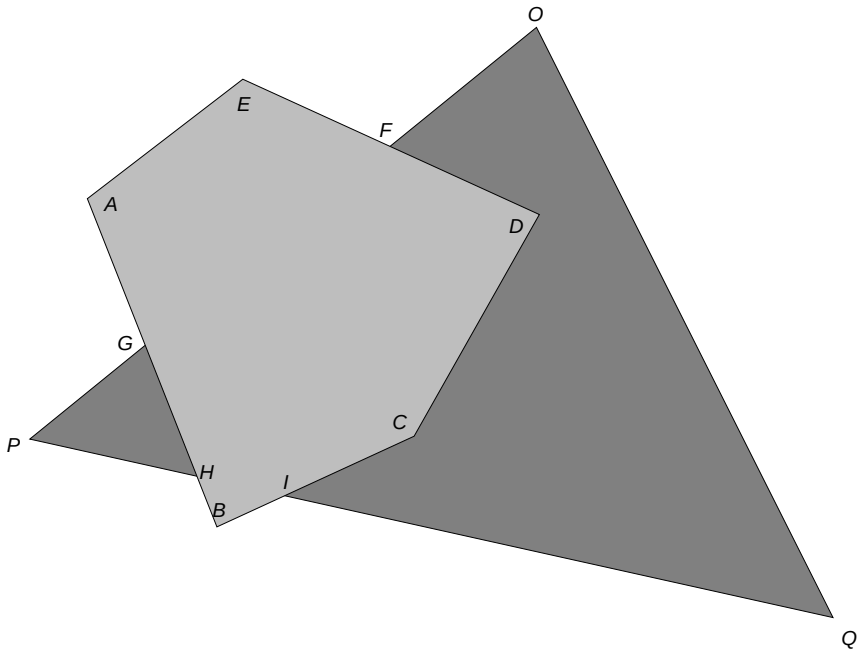


Figure 5.3: Visible Polygons

5.3 Constraint Generation

The purpose of constraint generation is to create the strong constraints according to the geometrical relationships among the polygon edges as discussed in Chapter 4. After the polygons are projected onto a $2D$ plane, they may overlap. What we are interested in are the visible polygon edges instead of the original polygon edges, which may be hidden. The polygons should be reconstructed so that only the visible parts of the polygons are included in the new collection of polygons. For example, two polygons $ABCDE$ and OPQ in Figure 5.2 are projected onto a plane, and they overlap. What concerns us is their visible polygon edges in polygons $ABCDE$, PHG and $OFDCIQ$ as in Figure 5.3. It is desirable to maintain the contrast across the edges DF, CD, IC, and GH.

5.3.1 A Modified Visible Surface Algorithm

Many visible surface algorithms have been published in the last two decades. However, we need an algorithm that outputs explicit polygonal representation. An algorithm proposed by Sechrest and Greenberg [SG82] and an algorithm by Nurmi have such a property [Nur85]. Both Sechrest and Greenberg's algorithm and Nurmi's algorithm are based on a sweeping paradigm. The sweeping approach has been mostly used for intersection problems, triangulation, and order problems [Meh84, page 147]. The sweeping paradigm was introduced by Shamos and Hoey to solve geometric intersection problems in 1976 [SH76]. Hamlin and Gear [HG77] used the sweeping method to solve the visible surface problem in 1977.

In Sechrest and Greenberg's algorithm, edges and vertices shared by more than one polygon are explicitly represented as shared, but the Nurmi's algorithm assumes that each edge only belongs to one polygon. We keep separate copies for the shared vertices and edges in the relative polygons because in our problem, though those shared vertices have the same $3D$ geometrical coordinates, they may have different colours. Sechrest and Greenberg's algorithm does not consider the cases of coincident edges[2] or coincident sweeping points[3], and Nurmi's algorithm does not consider the coincident edges either. Of course neither of them produce the constraints we desire. We use a modified algorithm based on these two algorithms to generate the strong constraints. This algorithm sweeps the polygons from top to bottom. It uses an Active Edge List (AEL) to record the status of the sweep at the current position of the sweeping line. The status of

[2] Two edges are coincident, if they have the same slope and have at least one point in common.

[3] the term sweeping point will be defined later.

the sweep is all information about the polygon visibility, polygon edge location (the sequence of active edges in AEL), and so on, above the sweep line which is relevant to solving the polygon visibility, polygon edge location below the sweep line. The sweeping procedure updates the AEL at sweeping points (to be defined later), outputs $2D$ vertices and visible segments to relevant polygons, and creates two and three variable constraints as defined in Equations 4.21, 4.22, and 4.24 during the process of sweeping. Then a polygon reconstruction function is called to reconstruct polygons from the visible segments.

We assume that the input polygons are all convex, though it is possible to produce concave polygons after the visible surface determination, and that no polygons penetrate each other.

Data Structures

We describe some data structures used to store the necessary information. They are defined precisely in Appendix C. In this subsection we only address a few interesting features of the data structures. The description is not necessarily sufficient for the implementation.

There are two types of vertices. One type is $3D$. It refers to the vertices before the visible surface determination. The other type is $2D$, which refers to the vertices output by the visible surface determination algorithm. The information stored in a $3D$ vertex is: the Euclidian coordinate $(x,\ y,\ z)$, the homogeneous coordinate $(hx,\ hy,\ hz,\ hw)$, and the initial colour in Space 0 $(r0,\ g0,\ b0)$. The information a $2D$ vertex stores are:

- the coordinate in $2D$ $(x,\ y)$,

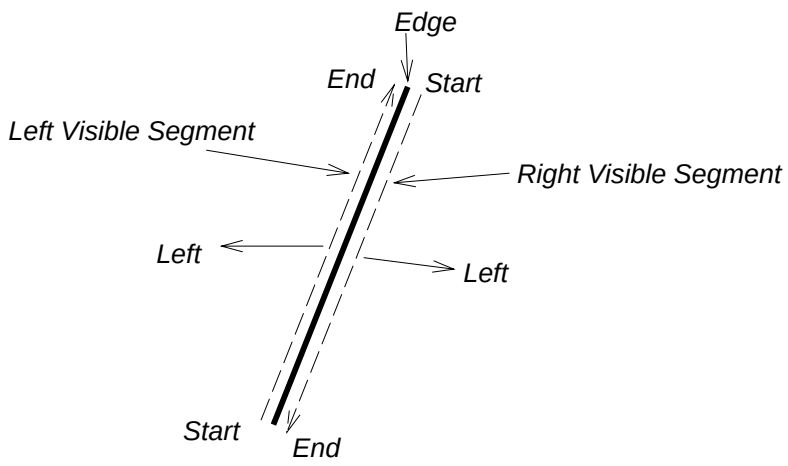


Figure 5.4: Directions of Visible Segments

- the identifying number for this vertex to be used in solving the BLLS,
- the colour in Space 0 $(r0,\ g0,\ b0)$,
- the colour after the first linear transformation $(r1,\ g1,\ b1)$, and
- all the HSV or RGB colours in other spaces during various stages of the rendering pipeline.

Two overlapping visible polygons are always separated by an edge. For an edge that is visible, a visible segment is added to the left shallowest[4] polygon, and another visible segment is added to the right shallowest polygon. A visible segment is a candidate for a polygon edge during the polygon reconstruction process (which will be addressed later). An edge has the following attributes:

- the pointer to its polygon,

[4] "Shallowest" means that a polygon is closest to the view point.

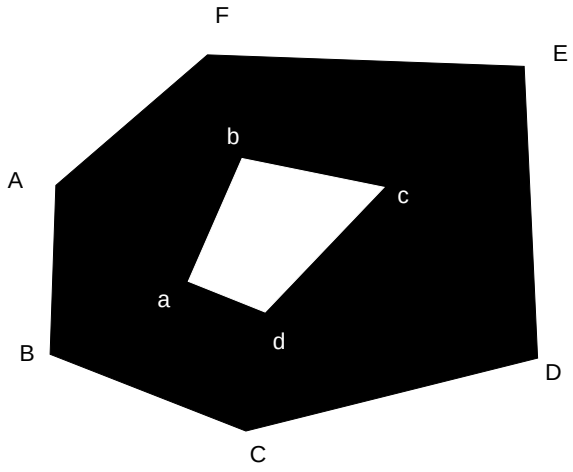


Figure 5.5: A Hole in A Polygon

- the pointers to the shallowest polygons to the left and right of the edge,
- its start and end $3D$ vertices,
- the pointers to the left and right visible segments,
- a pointer to a $2D$ vertex where this edge becomes visible, and
- a list of three variable constraints related to this edge to be completed.

A visible segment structure contains:

- the pointers to start and end $2D$ vertices,
- a pointer to the polygon this visible segment belongs to, and
- a thread flag denoting if this visible edge is a thread[5].

[5] A thread is a pseudo visible segment directed from a vertex to another. The contour the start vertex belongs to might be a hole of the contour the end vertex belongs to. See Appendix B for more detail.

An outer contour of a polygon has a counter-clockwise orientation, while a hole[6] has a clockwise orientation. Therefore, when we walk through a visible segment from its start vertex to its end vertex, the polygon to which this visible segment belongs is always to the left of the visible segment. That means the left visible segment of an edge starts at the lower $2D$ vertex, and ends at the upper $2D$ vertex; the right visible segment of an edge starts at the upper $2D$ vertex, and ends at the lower $2D$ vertex (See Figure 5.4).

A polygon structure has:

- a list of $3D$ vertices that form the contour of this polygon,
- a list of $2D$ vertices output for this polygon by the algorithm,
- a list of visible segments output for this polygon,
- the polygon equation, and
- a list of slave polygons that will be produced from this polygon during the polygon reconstruction.

The algorithm also maintains a few global data structures: a priority queue, an active edge list, and a list of strong constraints. A priority queue (PQ) consists of sweeping points (SP) sorted by Y and X. A sweep point is either a $3D$ vertex of the polygons or an intersection of the polygon edges in $2D$. An active edge list (AEL) holds the polygon edges intersecting the current sweeping line sorted by X. A sweeping line is a horizontal line passing through a sweeping point. An

[6] A hole is an interior contour of a polygon. For example, as shown in Figure 5.5, polygon $ABCDEF - abcd$ has outer contour $ABCDEF$, and interior contour *abcd*. *abcd* is a hole of this polygon.

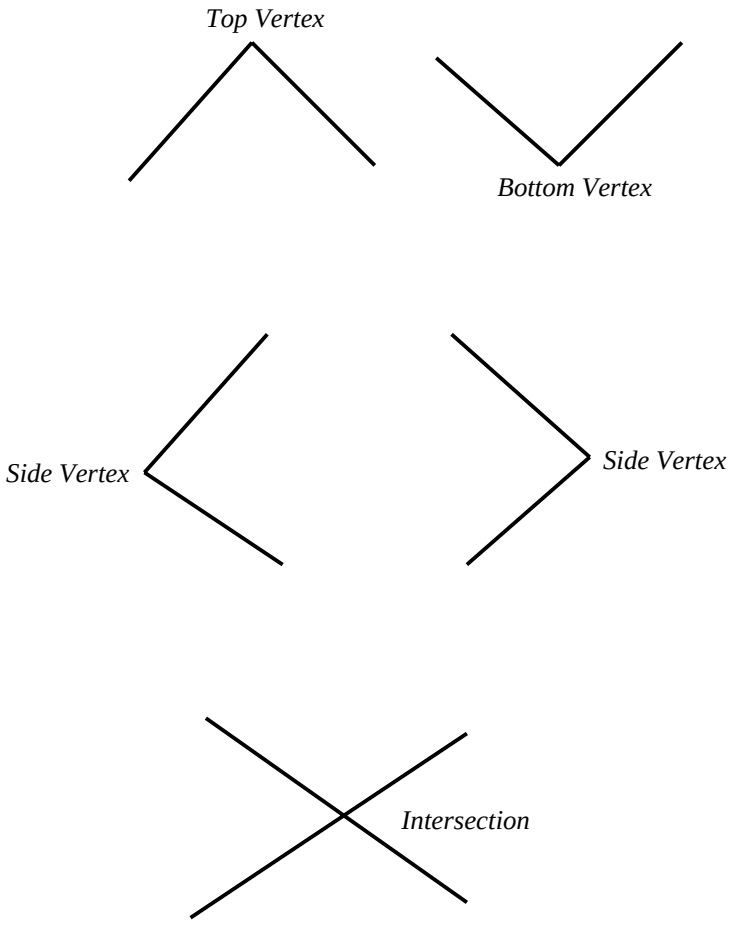


Figure 5.6: Sweeping Points

element in the active edge list has two components: a pointer to an edge and a pointer to a node list (NL). A node list is a list of pointers to polygons which lie in the span between this edge and the next non-coincident edge in the AEL. An NL is sorted by the polygons' depths at the sweeping point, and is used during the sweeping to find the shallowest polygon efficiently.

Sweeping

A sweeping point is either a $3D$ polygon vertex or an intersection point of polygon edges. A $3D$ vertex can be either a top vertex, bottom vertex, or side vertex (See Figure 5.6). Because two edges intersect only if they are next to each other in the AEL, an intersection point can be detected by checking if an edge e intersects its predecessor and / or successor when e is inserted into the AEL, and by checking if an edge e's predecessor intersects e's successor when e is removed from the AEL. The sweeping procedure halts at the sweeping points: all the polygon vertices and all the intersection points of the polygon edges. At these sweeping points, the sweeping procedure outputs $2D$ vertices and visible segments to relevant polygons so that the polygon reconstruction routine can reconstruct the polygons from these visible segments and $2D$ vertices. It also outputs strong constraints to the global list of strong constraints, which are to be solved by the constraint solver.

The implementation of the complete sweeping algorithm is rather complicated. In this section, we only give a rough description of this algorithm and provide a simple example, so that a reader may have some intuitive understanding of its operation. The detailed description of this algorithm is in Appendix B and the pseudo-code is in Appendix C.

	Polygon OPQ		Polygon $ABCDE$	
S.P.	$2D$ Vertices	Visible Segments	$2D$ vertices	Visible Segments
O	o	1,2		
E	o	1,2	e	3,4
F	o, f	1,2,5	e	3,4
A	o, f	1,2,5	e, a	3,4,6
D	o, f, d'	1,2,5,8	e, a, d	3,4,6,7
G	o, f, d', g	1,2,5,8,9,10	e, a, d	3,4,6,7
C	o, f, d', g, c'	1,2,5,8,9,10,12	e, a, d, c	3,4,6,7,11
P	o, f, d', g, c', p	1,2,5,8,9,10,12,13	e, a, d, c	3,4,6,7,11
H	o, f, d', g, c', p, h	1,2,5,8,9,10,12,13	e, a, d, c	3,4,6,7,11
I	o, f, d', g, c', p, h, i	1,2,5,8,9,10,12,13,14	e, a, d, c	3,4,6,7,11
B	o, f, d', g, c', p, h, i	1,2,5,8,9,10,12,13,14	e, a, d, c, b	3,4,6,7,11
Q	$o, f, d', g, c', p, h, i, q$	1,2,5,8,9,10,12,13,14	e, a, d, c, b	3,4,6,7,11

Table 5.2: A Progressive List of $2D$ Vertices And Visible Segments

S.P.	Strong Constraints
O	
E	
F	$\{f, e, ?\}$
A	$\{f, e, ?\}$
D	$\{f, e, d\}\{d, d'\}$
G	$\{f, e, d\}\{d, d'\}\{g, a, ?\}$
C	$\{f, e, d\}\{d, d'\}\{g, a, ?\}\{c, c'\}$
P	$\{f, e, d\}\{d, d'\}\{g, a, ?\}\{c, c'\}$
H	$\{f, e, d\}\{d, d'\}\{g, a, ?\}\{c, c'\}\{h, a, ?\}$
I	$\{f, e, d\}\{d, d'\}\{g, a, ?\}\{c, c'\}\{h, a, ?\}\{i, c, ?\}$
B	$\{f, e, d\}\{d, d'\}\{g, a, b\}\{c, c'\}\{h, a, b\}\{i, c, b\}$
Q	$\{f, e, d\}\{d, d'\}\{g, a, b\}\{c, c'\}\{h, a, b\}\{i, c, b\}$

Table 5.3: A Progressive List of Strong Constraints. In the table $\{x, y\}$means there is a two variable constraint between x and y; $\{x, y, z\}$means x, y, and z have a three variable constraint; ? means this vertex is not known yet.

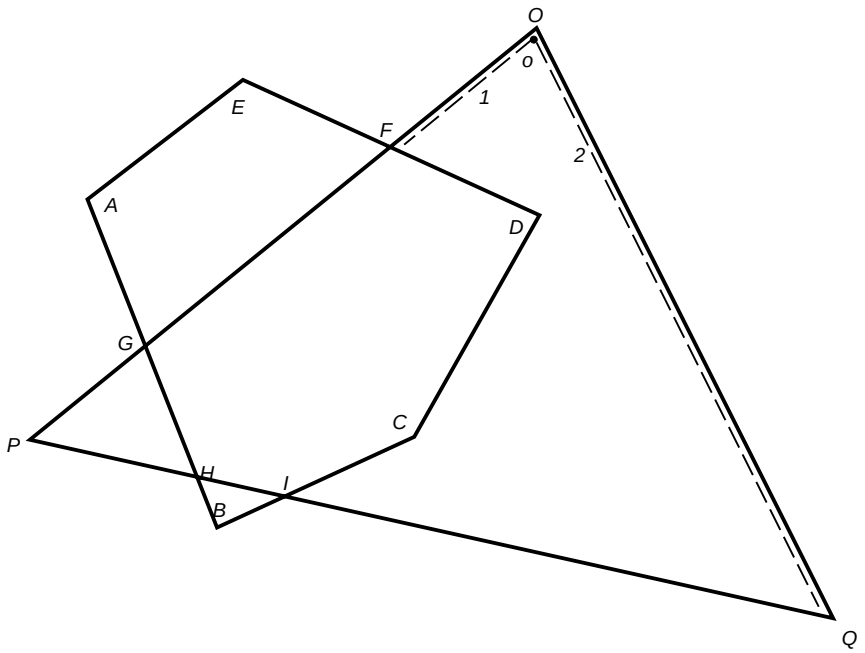


Figure 5.7: At Sweeping Point O

The theme of this sweeping algorithm is that when we handle an edge, we look to the left / right of the edge, then perform some actions based on the circumstances. At a SP, first, we deal with the edges above this sweeping point. Then, we handle the edges below it.

To an edge above the SP, if the SP is an intersection, we do not do anything because this edge will continue after the SP. If the SP is not an intersection, and if the edge has a left visible segment, we emit a $2D$ vertex to the polygon the visible segment belongs to, and close this visible segment by giving the last output vertex as its start / end vertex. If the edge has a right visible segment, similar operations are conducted. Two or three variable constraints are created in accordance with the geometrical relationship among the $2D$ vertices' corresponding visible segments. For an edge below this SP, if the edge is visible and does not have visible

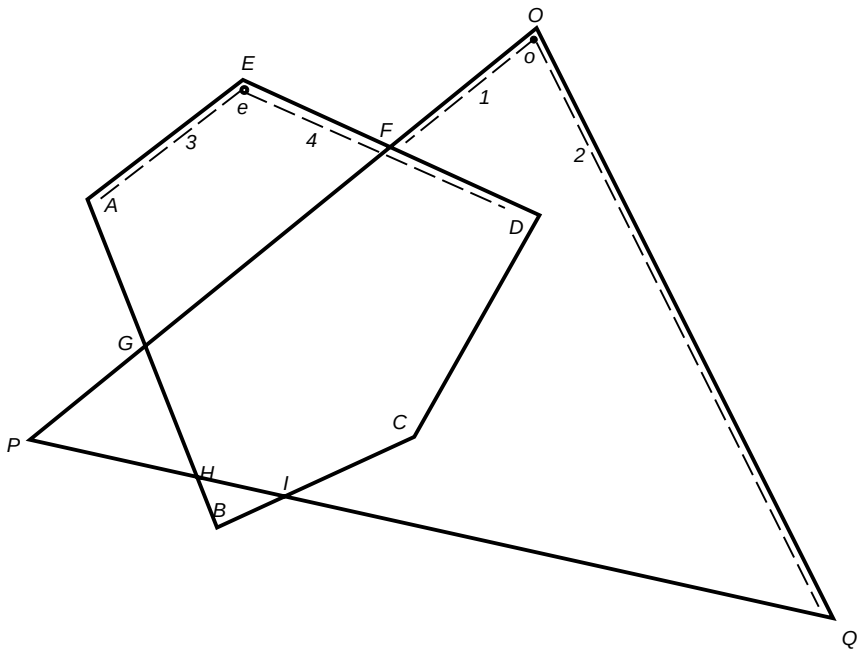


Figure 5.8: At Sweeping Point E

segment(s) yet, we output $2D$ vertices and visible segments to the shallowest polygons left and right of this edge. Strong constraints are also generated based on the geometrical relationships.

We will go through the sweeping process for the scene composed of polygons $ABCDE$ and OPQ as in Figure 5.3. The operations on relevant data structures are to be addressed. A progressive list of the $2D$ vertices and visible segments emitted during the sweeping procedure is in Table 5.2, while a progressive list of the strong constraints created during the sweeping is in Table 5.3.

From top to bottom, the first SP is O (Figure 5.7). Since there are no edges above O, we handle the edges below O immediately. At O, edge OP is visible, hence we output a $2D$ vertex o to OP's polygon (polygon OPQ). There is no polygon to the left of OP, so OP does not have a left visible segment. OP has

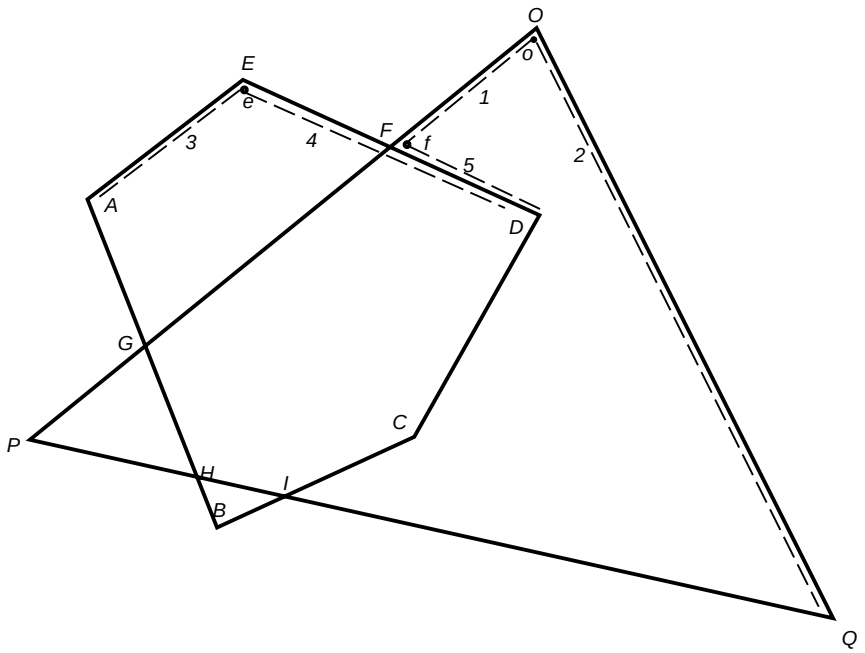


Figure 5.9: At Sweeping Point F

right visible segment 1, whose start vertex is o, and segment 1 is output to the shallowest polygon right of OP (polygon OPQ). Edge QO is also visible at O, so we want to output a $2D$ vertex to QO's polygon, but we find o already exists at O, thus we take it instead of outputting a $2D$ vertex again. QO's left visible segment 2, with o as its end vertex, is output to polygon OPQ. The sweeping point E is also a top vertex, and the edges EA and DE are dealt with in a similar fashion (See Figure 5.8).

The sweeping point F is an intersection point (Figure 5.9). The edges above this SP are DE and OP. We do not output anything because the edges will continue after F, but we must remember that DE is visible above F and the shallowest polygon left of DE is polygon $ABCDE$, and so on. We now deal with the edges below F. OP becomes invisible, so we output a $2D$ vertex to

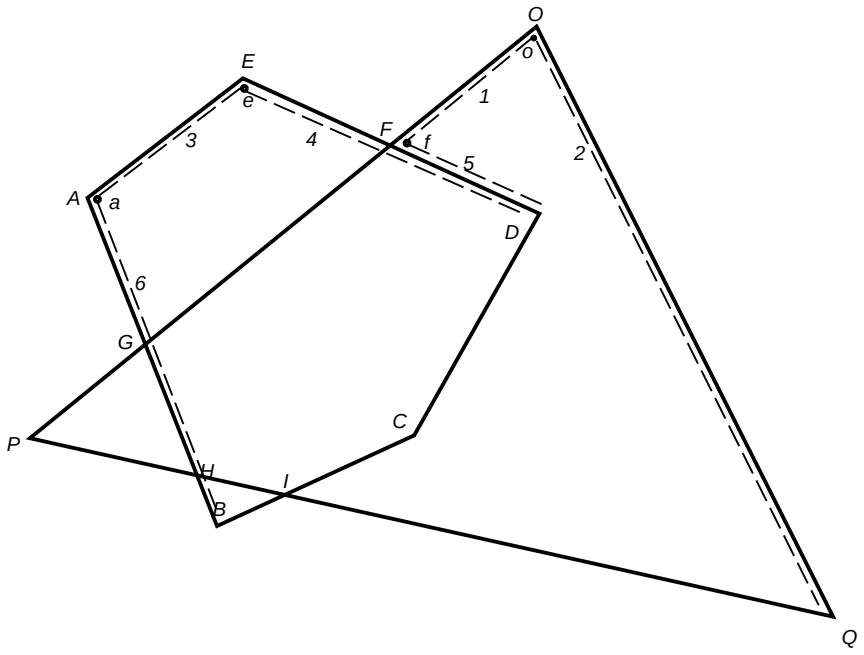


Figure 5.10: At Sweeping Point A

the shallowest polygon right of OP before F, i.e. we output f to polygon OPQ. Visible segment 1's end vertex is assigned to be f and segment 1 is closed. Edge DE is still visible, and the shallowest polygon left of DE is the same one as before the SP, thus we do not do anything to DE's left visible segment. The shallowest polygon right of DE (polygon OPQ) is not the same one as before F, so we want to output a $2D$ vertex to OPQ, but we find f exists at F. Therefore we just take f and set it to be the start vertex of the visible segment 5 which is output to polygon OPQ. As we discussed in Chapter 4, we want to retain the contrast across visible segments 5 and 4, hence we produce a three variable constraint among f, e, and the start vertex of segment 4, which has not been reached yet. The sweeping points G, H, and I are also intersection points. The edges corresponding to these SPs are treated in a similar way to the edges DE and OP. Figures 5.12, 5.15, and 5.16 depict their treatment respectively.

In Figure 5.10, the sweeping point A is a side vertex. We output a $2D$ vertex a to polygon $ABCDE$, and close visible segment 3 with a as its end vertex. A right visible segment 6 for edge AB is output to polygon $ABCDE$, and a is its start vertex. At the sweeping point D (See Figure 5.11), to edge above D, DE has both left and right visible segments, so we output $2D$ vertices d and d to the left and right polygons respectively, and close these visible segments by setting d as segment 4's start vertex and d' as segment 5's end vertex. A two variable constraint $\{d,\ d'\}$ is generated in order to maintain the contrast across visible segments 4 and 5. Then we handle the edges below D, visible segments 7 and 8 are created as edge CD's left and right visible segments. Segment 7's end vertex is d, while 8's start vertex is d'. The sweeping points C and P are also side vertices, they are dealt with in the same style, as shown in Figures 5.13 and 5.14.

Sweeping point B is a bottom vertex (Figure 5.17). It does not have edges

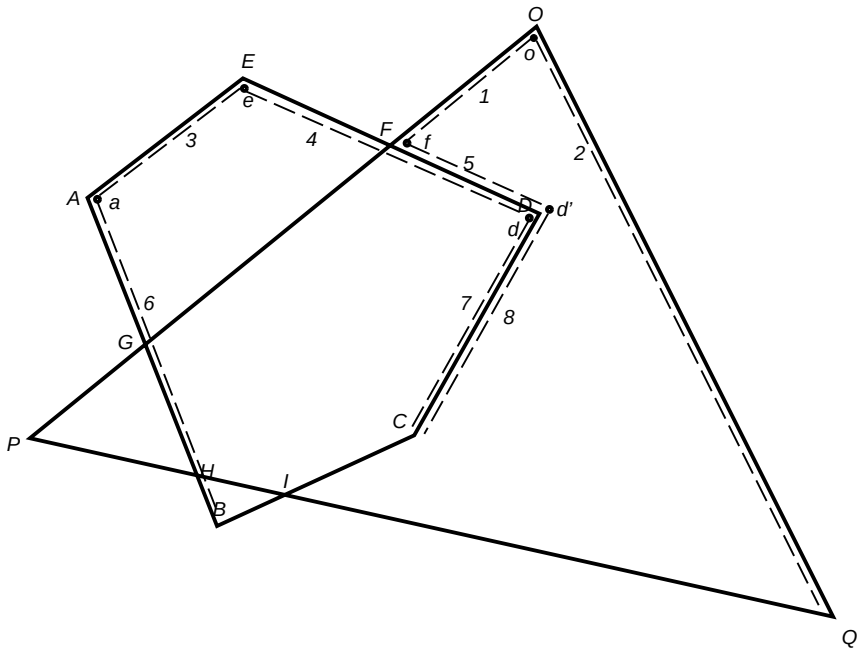


Figure 5.11: At Sweeping Point D

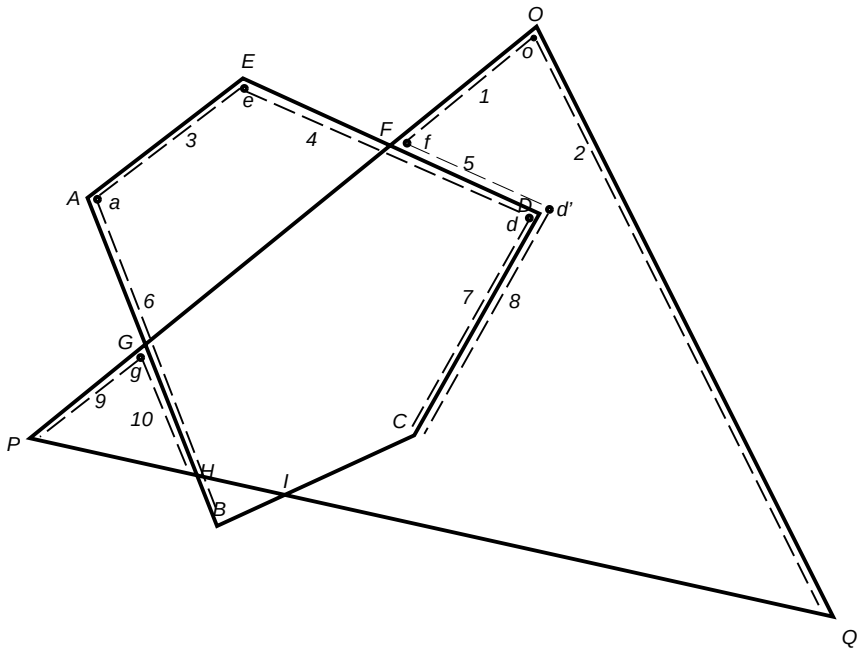


Figure 5.12: At Sweeping Point G

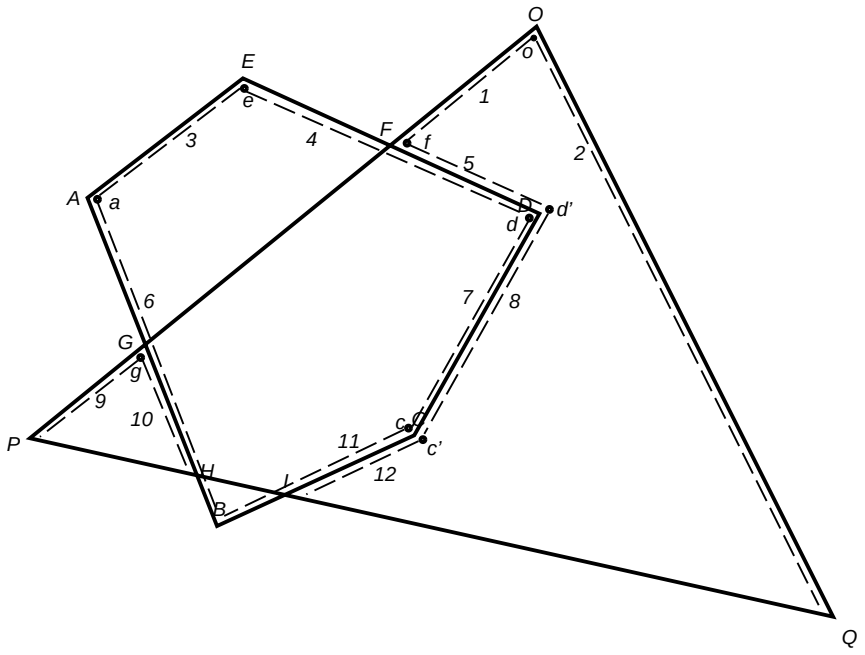


Figure 5.13: At Sweeping Point C

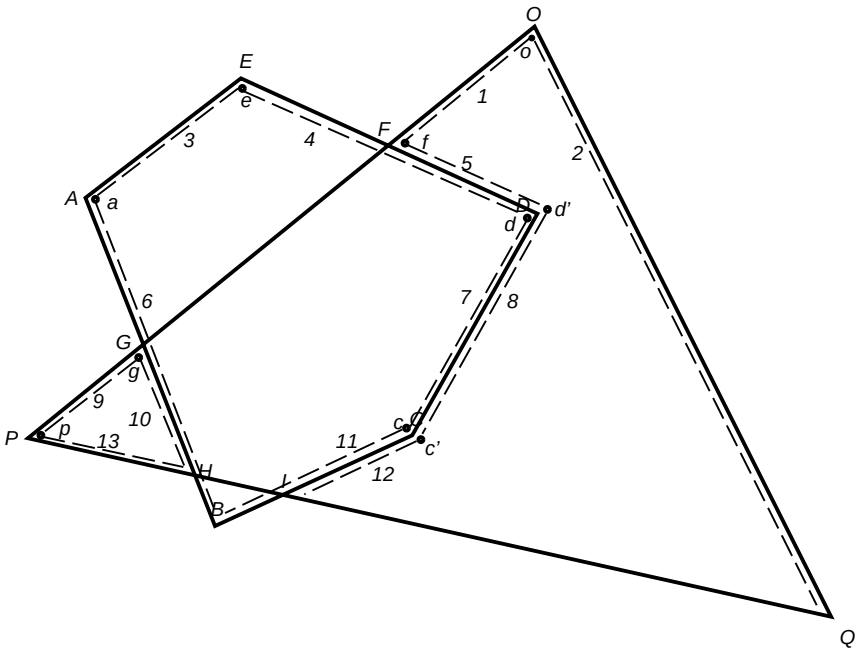


Figure 5.14: At Sweeping Point P

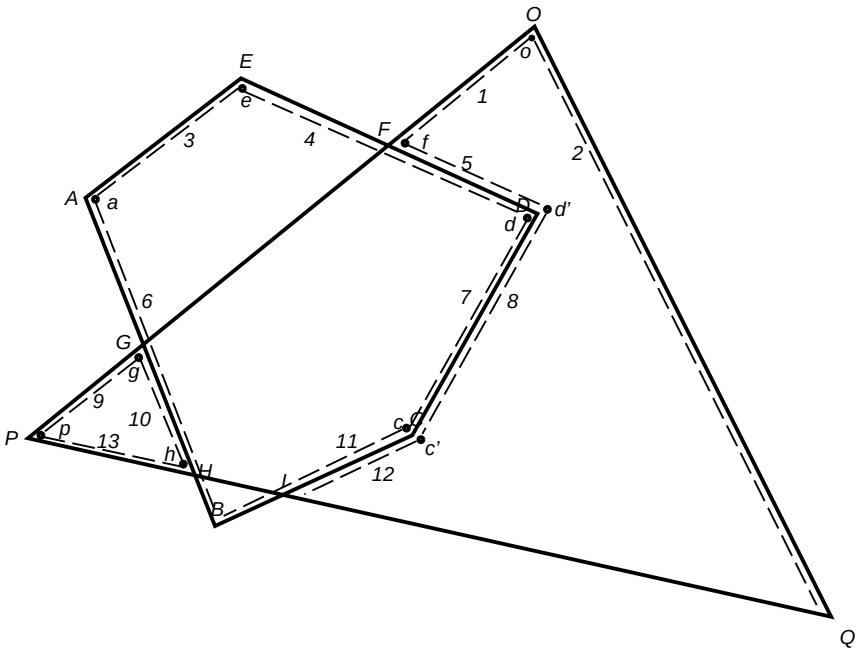


Figure 5.15: At Sweeping Point H

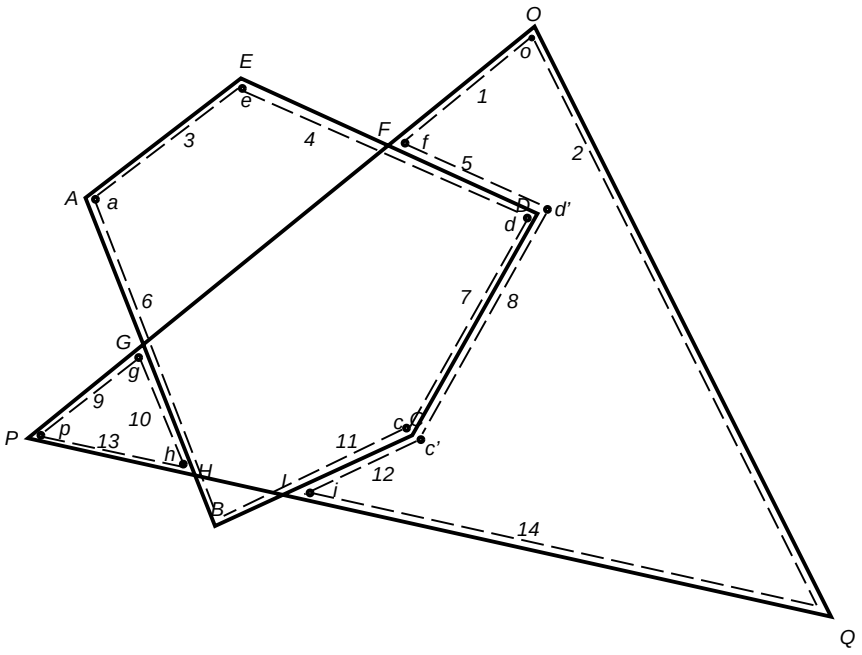


Figure 5.16: At Sweeping Point I

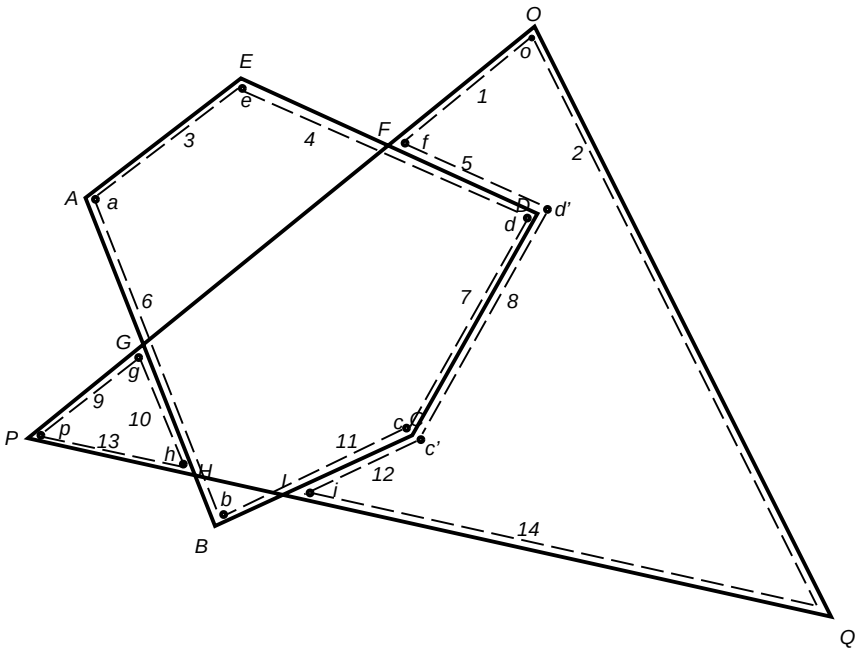


Figure 5.17: At Sweeping Point B

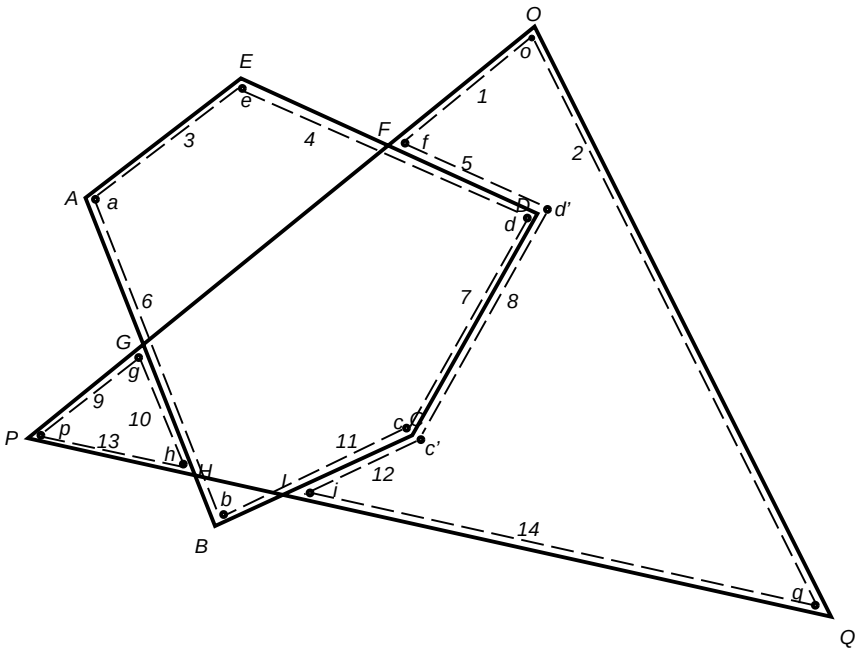


Figure 5.18: At Sweeping Point Q

below it. The above edge AB does not have left visible segment, but it has a right visible segment, so we output a $2D$ vertex b to the shallowest polygon right of AB (polygon $ABCDE$), and AB's right visible segment 6's end vertex is set to b. Later, we deal with the edge BC, and b is set to be the start vertex of edge BC's left visible segment (segment 11). The edges relevant to the sweeping point Q are handled in a similar approach, as shown in Figure 5.18.

In summary, the sweeping procedure has output $2D$ vertices o, f, d', g, c', p, h, i, q and visible segments 1, 2, 5, 8, 9, 10, 12, 13, 14 to polygon OPQ. It has as well output $2D$ vertices e, a, d, c, b and visible segments 3, 4, 6, 7, 11 to polygon $ABCDE$. The polygon reconstruction function will reconstruct new polygons from these $2D$ vertex lists and visible segment lists. Strong constraints $\{f, e, d\}$, $\{d, d'\}$, $\{g, a, b\}$, $\{c, c'\}$, $\{h, a, b\}$, and $\{i, c, b\}$ have been imposed, and they will be solved in the constraint solving block.

Appendix B shows that the running time for this sweeping algorithm is $O((n+k)n)$, where n is the number of vertices and k is the number of edge intersections in a scene respectively.

Polygon Reconstruction

After the sweeping procedure is finished, every polygon has a list of $2D$ vertices and visible segments. The next step is to reconstruct polygons from these visible segments. Each polygon may yield a few disjoint polygons, and each new polygon (slave polygon) may have holes. We will discuss how to reconstruct polygons from the visible segments for a polygon. A visible segment becomes an edge of a new polygon.

It is easy to show that during the sweeping the first visible segment for an

Visible Segment	Starting $2D$ Vertex	Ending $2D$ Vertex
1	o	f
2	q	o
3	e	a
4	d	e
5	f	d'
6	a	b
7	c	d
8	d'	c'
9	g	p
10	h	g
11	b	c
12	c'	i
13	p	h
14	i	q

Table 5.4: Visible Segments and their Start/End Vertices

outer contour of a new polygon is always output earlier than the first visible segment for a hole of this new polygon. We start at the beginning of the list of visible segments, and pick a visible segment, whose start vertex is the same as the previous visible segment's end vertex, as the next edge, until we reach a visible segment whose end vertex is the same as the first visible segment's start vertex. These visible segments make up the outer contour of a new polygon. If there are any threads directly or indirectly[7] ending at vertices of this contour, then for each thread we find the visible segment with a start vertex the same as this thread's start vertex, and repeat the process of constructing the outer contour. The resulting contours are the holes of this new polygon. If there are still visible segments remaining in the list, we repeat the whole process and create other new polygons until the list is empty.

For instance, Table 5.4 lists the visible segments and their corresponding start and end vertices produced in the example illustrated in the sweeping subsubsection. The original polygon OPQ has visible segment list {1, 2, 5, 8, 9, 10, 12, 13, 14}. The first visible segment starts at o and ends at f. We look for the visible segment starting at f, and it is segment 5, so segment 5 becomes the second edge of the polygon we are constructing. So on and so forth, we will reach visible segment 2, which ends at o. Thus we have constructed an outer contour {1, 5, 8, 12, 14, 2} for a new polygon $ofd'c'iq$. Because we do not see any threads ending at the visible segments for this contour, this new polygon does not have any hole. Similarly, the remaining visible segments {9, 10, 13} also form a new polygon gph.

After the constraint generation, the first linear transformation, RGB to HSV transformation, and the second linear transformation (as described in Chapter 4)

[7] "Indirectly ending at a vertex of this contour" refers to a thread ending at a hole which in turn has a thread directly or indirectly ending at a vertex of this contour.

are performed for all the vertices of the new polygons.

5.4 Constraint Solving

From Chapter 4, we know that the constraint hierarchy can be formalized as a bounded linear least squares problem. Bounded constraint is the simplest linear constraint. The NAG package [NAG91] has a function E04NCF, which solves linearly constrained linear least squares problems and quadratic programming problems. Therefore, we use the NAG library to solve our constraint satisfaction problem. The University of Waterloo only has a site license for the Fortran binding of the NAG package. Fortunately, it is not hard to call a Fortran function from C or to call a C function from Fortran except that some attention must be spent on function name conventions and function parameter passing. Although Fortran does not have dynamic memory management, we can dynamically allocate a block of memory in C, then pass it to Fortran.

The BLLS can be written as:

$$\begin{aligned} &Minimize && \|AX - B\|_2^2 \\ &subject\ to && L \leq X \leq U \end{aligned} \tag{5.8}$$

where A is the least square matrix, B is the vector of observations, L is the vector of lower bounds, U is the vector of upper bounds, and X is the vector of variables.

The constraint solver first allocates the matrix A, and vectors B, L, U, and X based on the problem size and the number of strong constraints created. The solver walks through the internal data structures which store the strong constraints produced in the constraint generation block. When it meets a strong

constraint in the format of Equation 4.21, the luminances of two vertices involved have the relationship $V_j - cV_k = 0$, where j and k are the identifying numbers for these two vertices separately, and c is the contrast in Space 0. It assigns $A[i][j] = \Phi$, $A[i][k] = -c\Phi$, and $B[i] = 0$, where Φ is the weight for the strong constraints, and i is a count initialized to be 0 and i increases by 1 whenever one row of A is set up. When the solver comes into a strong constraint like Equation 4.22, the luminances of three vertices have the relationship $V_j - c(1-\beta)V_k - c\beta V_l = 0$, where l is also an identifying number for a vertex. It sets $A[i][j] = \Phi$, $A[i][k] = -c(1-\beta)\Phi$, $A[i][l] = -c\beta\Phi$, and $B[i] = 0$. When it sees a strong constraint in the format of Equation 4.24, three vertices' luminances have the relationship $-cV_j + (1-\beta)V_k + \beta V_l = 0$. It assigns $A[i][j] = -c\Phi$, $A[i][k] = (1-\beta)\Phi$, $A[i][l] = \beta\Phi$, and $B[i] = 0$.

The solver now sets up the rows in A and B for the medium constraints. For every vertex and every polygon, suppose the vertex's identifying number is j and the identifying number for the vertex next to it in the same polygon is k, the solver assigns $A[i][j] = \Psi$, $A[i][k] = -\Psi$, and $B[i] = 0$, where Ψ is the weight for the medium constraints. To set up the rows for the weak constraints, for each vertex in each polygon, assuming its identifying number is j, and its luminance after the second linear transformation is V_{j_0}, the solver lets $A[i][j] = \Theta$, and $B[i] = V_{j_0}\Theta$, where Θ is the weight for weak constraint. The NAG function E04NCF requires the caller to supply an initial estimate of the solution. We use the luminance in Space 3 as the initial estimate, thus the solver assigns $X[j] = V_{j_0}$. In our problem, all the luminances have the same lower bounds and upper bounds, hence the solver sets $L[j] = LowerBound$ and $U[j] = 1$, where the meaning of $LowerBound$ was explained in Chapter 4.

After all the relevant matrices and vectors have been set up, the solver calls

the NAG function E04NCF. When E04NCF returns, the solver sets the luminance V for each vertex in the internal data structures to be the value of corresponding $X[j]$.

Because the NAG function E04NCF uses an iterative method to solve the BLLS problem, the expected running time is not reflected by the worst case running time. In practice, with our problem size, we have prompt response. When the problem size increases, other constraint solving methods may need to be investigated.

The HSV to RGB transformation as described in Chapter 4 is also performed in this block. H and S are the old H and S, but V is the new one produced by the constraint solver. The third linear transformation is carried out here as well. Another task performed in this block is window to view-port mapping. The window is $[-1,\ 1] \times [-1,\ 1]$, and the view-port is $[0,\ MaxX] \times [0,\ MaxY]$. A vertex $(x,\ y, z)$'s x and y are mapped by Equation 5.9, but z is not mapped because z is not meaningful anymore.

$$\begin{aligned} New\ x &= \frac{MaxX}{2}(x\ +\ 1) \\ New\ y &= \frac{MaxY}{2}(y\ +\ 1) \end{aligned} \tag{5.9}$$

5.5 2D Rendering

After the constraint generation has been performed, the hidden surface removal has also been finished. We have a list of polygons that do not overlap in $2D$. Therefore, it does not make any difference which polygon we render before the other. The $2D$ renderer scan converts polygons one by one. A polygon may be

concave or may have holes. The scan conversion algorithm we used is based on an algorithm by Heckbert [Gla90, pages 87-91 and 681-684].

The scan converter sorts all the vertices by their Y coordinates and maintains an active edge list. An active edge is an edge which intersects the present scan line. When the scan converter goes to the next scan line, it updates the active edge list efficiently according to the sorted vertex list. At each scan line, every other horizontal segment of the scan line separated by the polygon edges should be filled. In each segment to be filled, the R, G, and B values are linearly interpolated from the R, G, and B values at the intersections of the scan line with the polygon edges, which in turn are the linear interpolations of the R, G, and B values at polygon vertices.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Display hardware contrast limitations were addressed in depth in this thesis, along with related properties of the human visual system. Based on the display device restriction and properties of human visual system, a new constraint-based rendering framework was proposed, and a modified rendering pipeline was implemented based on this framework.

The prototype system has rendered some scenes with high dynamic ranges. Comparing with the images rendered by a conventional rendering method (called RMA) without considering limitation on the display device dynamic range and by a rendering method (called RMB) with simple linear transformation (which transforms the high dynamic range into the low dynamic range as discussed in Chapter 4), the image rendered by the proposed constraint-based rendering method (called RMC) has better image quality and presents more information to a viewer.

An experiment was conducted by measuring luminances in a real world high dynamic range scene, whose highest luminance was 1200 and lowest luminance was 2, with a radiometer, taking photographs on the scene with a camera, and rendering the scene by RMA, RMB, and RMC based on the data collected by the radiometer. In the photographs of the scene, either the dark part had no perceivable detail or the bright part was washed out. The image produced by RMA has the worst quality, and the image created by RMC has the best image quality. We also found that the weights for the constraint hierarchy influence the image quality heavily. In fact, it is not necessary that the weight for strong constraints is the largest and the weight for weak constraints is the smallest. We did not explore the optimal weight combination for a scene.

The contributions of this thesis can be summarized as following:

- *The research offers a way to overcome the display contrast limitation.* Display hardwares always have their limitations in today's technology, and they will still have in the near future. To render scenes with high dynamic ranges onto the display devices with lower dynamic ranges, the constraint-based technique developed in this thesis will preserve the visual contrast and thus we expect to enhance the perceived image quality.

- *The constraint-based model developed is useful for device independent image representation.* Different devices have different limitations. The constraint-based rendering model developed in this thesis is not only valuable in solving the problems caused by CRT contrast restriction, but also useful for device independent image representation. In a constraint-based device independent image representation scheme, the attributes of a scene are expressed as a set of constraints among the attributes. A constraint satisfier dedicated to a

particular device solves the constraints and generates the explicit attribute values to be displayed on the specific device.

- *Constraint-based approach is also useful for geometrical relations.* Although this thesis dealt with the constraints on colour, the principles of constraint-based image representation should also apply to geometrical relationships. For instance, in his master's thesis, Schlueter worked on perceptual synchronization in window systems [Sch90]. If the window edges of two adjacent windows are colinear, it causes ambiguity to a user. We can impose a constraint that no adjacent window edge are allowed to be colinear. A constraint satisfier will then assign the coordinates to window edges so that no adjacent windows' edges are colinear. Similar considerations seem to be important to human renderers, from draughtsmen to artists.

6.2 Future Work

This thesis has explored the display device contrast limitations and opened avenues for future research. Some further research topics are:

- Our constraint-based rendering model only works for flat and Gouraud shadings. A natural extension to this model is to make it also work for Phong shading, ray tracing, and radiosity.

- If we slowly vary the luminance, a viewer is not able to detect it. How much luminance variation within a unit distance is considered as slow? We need to add this quantitative measurement into the constraints.

- As stated in the conclusion, weights for the constraint hierarchy affect image quality heavily. Automatic determination of optimal weights for a scene needs to be systemically explored.

- In the constraint formalization of Chapter 4 and the constraint solving of Chapter 5, we can see that the least squares matrix of BLLS is very sparse. In general, BLLS is computationally expensive to solve. Can we take advantage of the sparse structure of this matrix to solve our BLLS more efficiently? Another minimization principle might have been to minimize absolute values. This is directly convertable to a linear programming problem for which sparsity handling is well known. We need to know which minimization principle, minimizing the summation of squares or minimizing the absolute values, results better visual effect.

- As stated in the previous section, we can develop a constraint-based device independent image representation scheme based on the principles derived in this thesis.

- In constraint generation as described in Chapter 5, we impose constraints based on the $2D$ polygon geometrical relationships. The calculation of the geometrical relationship is sometimes numerically sensitive, and thus robust geometrical algorithms are needed.

- If we apply the rendering techniques to animation, how can we take advantage of temporal coherence to make constraint solution efficient?

Appendix A

Transformation Matrices

A.1 Matrices of Geometrical Transformations

The matrix of translation by $(x,\ y,\ z)$:

$$T_{(x,y,z)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix} \tag{A.1}$$

The matrices of rotation about an axis X, Y, or Z by an angle θ:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.3)}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.4)}$$

The matrix of scaling by $(S_x,\ S_y,\ S_z)$:

$$S_{(x,y,z)} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.5)}$$

A.2 Matrices of Viewing Transformations

A.2.1 Projection in Canonical Form

Figure A.1 is a cross-section of the viewing frustum which goes through the eye point and a point $(x,\ y,\ z)$, and is perpendicular to the XZ plane. From the figure, we have:

$$\frac{y'}{y} = \frac{|EP|}{|EQ|} \quad \text{(A.6)}$$

It is easy to show that $\frac{|EP|}{|EQ|} = \frac{n}{-z}$. Thus, we have:

$$\frac{y'}{y} = \frac{-n}{z} \quad \text{(A.7)}$$

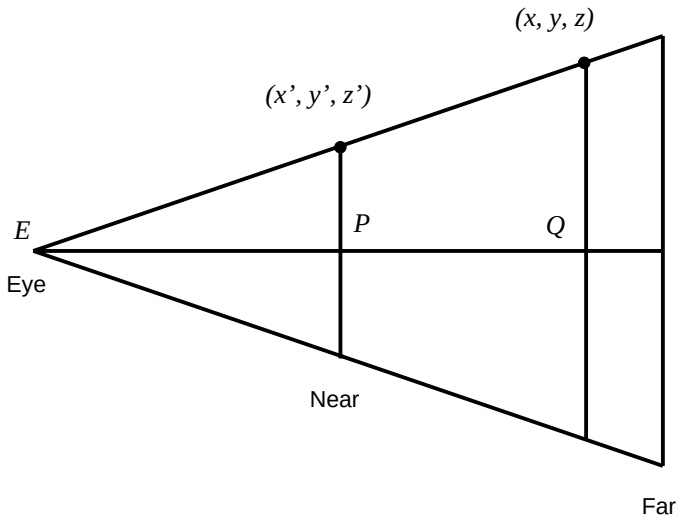


Figure A.1: Perspective Projection

Following the same reasoning, we have:

$$\frac{x'}{x} = \frac{-n}{z} \tag{A.8}$$

We now only do not have the formulae for calculating z'. After the $3D$ points are projected onto a $2D$ plane, the purpose for the Z coordinates is to preserve the depth information. According to Equations A.7 and A.8, if we represent the projection as a 4×4 matrix, only two entries in the matrix have not been determined:

$$(x'',\ y'',\ z'',\ w'') = (x, y, z, 1)P \tag{A.9}$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \gamma & -\frac{1}{n} \\ 0 & 0 & \delta & 0 \end{bmatrix} \tag{A.10}$$

where $x' = \frac{x''}{w''}$, $y' = \frac{y''}{w''}$, and $z' = \frac{z''}{w''}$. From these two equations, $z' = \frac{\gamma z + \delta}{-\frac{z}{n}}$. We can see in subsection 5.2.3 that it is desired to have $z' \in [0, 1]$. It is reasonable to let $z' = 1$ when $z = -f$, and $z' = 0$ when $z = -n$:

$$\frac{\gamma(-f) + \delta}{\frac{f}{n}} = 1 \tag{A.11}$$

$$\gamma(-n) + \delta = 0 \tag{A.12}$$

Solving the system of Equations A.11 and A.12, we obtain $\gamma = \frac{f}{n(n-f)}$ and $\delta = \frac{f}{n-f}$.

For the convenience of doing clipping, we map $x' = n\tan(\frac{\alpha}{2})$ to 1, $x' = -n\tan(\frac{\alpha}{2})$ to -1, $y' = \frac{n\tan(\frac{\alpha}{2})}{r}$ to 1, and $y' = -\frac{n\tan(\frac{\alpha}{2})}{r}$ to -1. As a result, P is revised to be:

$$P = \begin{bmatrix} \frac{\cot(\frac{\alpha}{2})}{n} & 0 & 0 & 0 \\ 0 & \frac{r\cot(\frac{\alpha}{2})}{n} & 0 & 0 \\ 0 & 0 & \frac{f}{n(n-f)} & -\frac{1}{n} \\ 0 & 0 & \frac{f}{n-f} & 0 \end{bmatrix} \tag{A.13}$$

A.2.2 Projection in Arbitrary Form

To the projection in arbitrary form, we transform it into the canonical form. To do so, we first translate the eye point to the origin $(0,\ 0,\ 0)$, then we rotate the sight vector into $-Z$ direction and the up vector into the Y direction.

Suppose the sight vector is $(S_x,\ S_y,\ S_z)$, the matrix to rotate the sight vector to $-Z$ is:

$$R_s = R_y(\Theta_y) \times R_x(\Theta_x) \tag{A.14}$$

where:

$$\sin(\Theta_x) = \frac{-S_y}{\sqrt{S_x^2 + S_y^2 + S_z^2}} \tag{A.15}$$

$$\cos(\Theta_x) = \frac{\sqrt{S_x^2 + S_z^2}}{\sqrt{S_x^2 + S_y^2 + S_z^2}} \tag{A.16}$$

$$\sin(\Theta_y) = \frac{S_x}{\sqrt{S_x^2 + S_z^2}} \tag{A.17}$$

$$\cos(\Theta_y) = \frac{-S_z}{\sqrt{S_x^2 + S_z^2}} \tag{A.18}$$

Suppose the up vector is $(U_x,\ U_y,\ U_z)$, after these rotations, the up vector becomes:

$$(U_x{}',\ U_y{}',\ U_z{}') = (U_x,\ U_y,\ U_z)R_s \tag{A.19}$$

Now we want to align the up vector with Y. Because the up vector is perpendicular to the sight vector, we only need to rotate it into the $+Y$ half plane of the $Y - Z$ plane.

$$R_u = R_z(\Theta_z) \tag{A.20}$$

where:

$$\Theta_z = \arctan(\frac{U_x{}'}{U_y{}'}) \tag{A.21}$$

Suppose, the eye point is $(E_x,\ E_y,\ E_z)$, the viewing transformation matrix is:

$$View = T(-E_x, -E_y, -E_z) \times R_s \times R_u \tag{A.22}$$

Therefore, the projection matrix in the arbitrary form is $View \times P$

Appendix B

Sweeping for Constraint Generation

The algorithm stops at all the vertices and all the intersections of the polygon edges from top to bottom and performs various actions at these sweeping points based on different circumstances.

Before the start of the sweeping, all the edge structures are created and initialized. Because the sweeping algorithm will assume no horizontal edges, all the $3D$ vertices' XY coordinates are rotated by an angle θ if there exists a horizontal edge. The θ is chosen so that no edge becomes horizontal after the rotation. After the sweeping, all the new output $2D$ vertices are rotated by angle $-\theta$ to offset the first rotation. All the vertices are classified to be certain types as defined in Figure 5.6 and inserted into the priority queue. The algorithm also computes the plane equations for all the polygons before the sweeping starts.

Overall, sweeping is an iterative procedure. At each iteration, it deletes the first sweeping point(s) from the priority queue, and conducts operations until the priority queue is empty. If it detects a new intersection point, the point is inserted into the priority queue. Because two edges intersect only if they are next

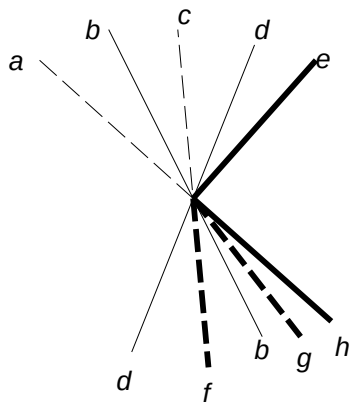


Figure B.1: Sweeping Points and Their Corresponding Edges

to each other in the AEL, an intersection point can be detected by checking if an edge e intersects its predecessor and/or successor when e is inserted into the AEL, and by checking if an edge e's predecessor intersects e's successor when e is removed from the AEL. It is possible that several sweeping points overlap in $2D$, i.e. they have the same XY coordinates. These sweeping points should be handled at the same time, so the sweeper deletes all the sweeping points with the same XY coordinates from the top of the priority queue in one iteration. In the following few paragraphs, the phrase "these sweeping points" also includes the case that there is only one current sweeping point.

Some edges corresponding to the current sweeping points end at these SPs, while other edges will continue after these SPs or start from them. For example in Figure B.1, edges a and c forms a bottom vertex; edges b and d intersect at this SP; edges e and h correspond to a side vertex; edges f and g forms a top vertex. Edges a, c, and e end at these SPs, while edges d, b, f, g, and h continue/start at these SPs. Above these SPs, from left to right, the edges are a, b, c, d, and e, but below the SPs, from left to right, the edges are d, f, b, g, and h. We want

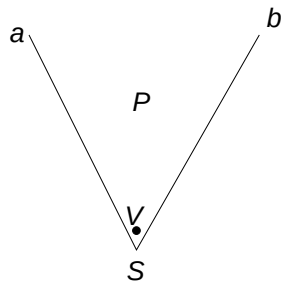


Figure B.2: Output a $2D$ Vertex

to process the edges corresponding to the current SPs from left to right. Thus, we have two lists of edges related to the current SPs, one for the edges above the sweeping line and another for the edges below the sweeping line. They are sorted from left to right as the previous example. We first handle the edges in the above edge list, and then deal with the edges in the below edge list.

Two edges are coincident if they have the same slope and have at least one point in common. It is possible that several polygon edges coincide. All the coincident edges are consecutive in the AEL, the above edge list, and the below edge list. To make the computation of the NLs and the handling of visible segments easy, the edges which are coincident to each other are sorted such that the edges whose polygons are to their left are placed before the edges whose polygons are to their right. The edges whose polygons are to their left are sorted such that the edges farther from the view point are placed before the shallower edges. Symmetrically, the edges whose polygons are to their right are sorted such that the shallower edges are placed before the edges farther from the view point. As a result, the edge whose polygon is the shallowest and is to the left of the coincident edges is placed just before the edge whose polygon is the shallowest and is to the right of the coincident edges.

To output a $2D$ vertex to a polygon at a SP, we add a $2D$ vertex into that polygon's list of $2D$ vertices. The XY coordinates for this $2D$ vertex are assigned to be the XY coordinates of the current SP. The RGB colour for this $2D$ vertex is set to the colour at this point when the polygon is rendered by the linear interpolating method. If the SP is a $3D$ vertex of the polygon where we wish to output a $2D$ vertex, the $2D$ vertex just inherits its colour from that $3D$ vertex. Otherwise the problem becomes: given a point (x, y) in a polygon and the colours of all the polygon vertices, which colour should this point have? Because we assume the input polygon is a convex polygon, and (x, y) is in the polygon, a line $Y = y$ intersects two polygon edges. We find these two edges by searching through all the edges in this polygon. The colour of an intersection point between one of these edges and the line $Y = y$ is the linear interpolation of the colours of its edge start / end vertices. The colour of (x, y) is then the linear interpolation of these two intersections' colours[1].

It is possible that when we want to output a $2D$ vertex to a polygon, the polygon already has a $2D$ vertex at the present SP. For example, in Figure B.2, when we deal with the edge b, and want to output a $2D$ vertex to the polygon P at the SP S, a $2D$ vertex V in fact has already been output when we handled the edge a. Therefore, in this section, when we say "output a $2D$ vertex to a polygon", we mean to output a $2D$ vertex to a polygon if that polygon does not have a $2D$ vertex at the current SPs yet (a $2D$ vertex with the XY coordinates the same as these SPs'). If the polygon already has one, take the existing one last output.

[1] If the vertex colours are not coplanar, linear interpolation may cause inconsistency [Bli92]. However, as discussed in Chapter 4, our model assumes that the variation of colour within any polygon is small, so that linear interpolation is acceptable

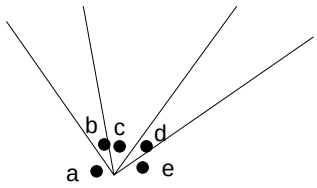


Figure B.3: 2 Variable Constraints Corresponding to "Above" Edges

B.1 Handling the Edges Above the SPs

For each edge e in the "above" list, if the SP corresponding to e is an intersection, we do not do anything except store the pointers to the shallowest polygons to the left and right of e into e's data structure to be used when we deal with e as a below edge. Otherwise, the SP corresponding to e is not an intersection. If e has a left visible segment, a $2D$ vertex is output to the shallowest polygon to e's left. The left visible segment is closed by giving this $2D$ vertex as its start vertex. If e's polygon is to the left of e, the start and end vertices of the visible segment might be the V_3 and V_4 of a three-variable constraint as in Equations 4.22 or 4.24. As we will see later, when we generate a three-variable constraint, only two vertices are known and thus specified. Hence we complete all the three-variable constraints related to this visible segment by specifying the third vertex of the constraints as the $2D$ vertex at this SP. If the SP corresponding to e is a bottom vertex, it is possible that the left visible segment is part of a hole. If it is part of a hole, the hole can only be a hole of the polygon to which the nearest visible segment to the left belongs. For the convenience of polygon reconstruction, we output a pseudo visible segment, named *thread*, from this $2D$ vertex to the top vertex of the nearest visible segment to the left. If e has a right visible segment, similar processing is also performed except no thread is considered.

We need to create two-variable strong constraints among the $2D$ vertices relevant to these "above" edges because we wish to retain the contrast across their visible segments. For example, in Figure B.3, vertices a and b, b and c, c and d, and d and e have two-variable strong constraints respectively. In the figure, for illustration purposes, these vertices do not coincide. In fact, they all coincide with the SP except that they are in different polygons.

All the edges in the above edge list are then removed from the AEL regardless of whether they will continue or not after the current SPs. When we delete an edge e from the AEL, we check if the edge before e and the edge after e in the AEL intersect. If they do, the intersection is inserted into the PQ.

B.2 Handling the Edges Below the SPs

All the edges in the below edge list are inserted into the AEL. The node list for an edge e is first copied from its predecessor in the AEL. If e's polygon is to the left of e, e's polygon is removed from its NL. Otherwise, if e's polygon is to the right of e, e's polygon is added to its NL. To ensure the node lists are correct for the edges that coincide, those coincident edges' NLs are computed together. We first insert the first one of the coincident edges into the AEL. The NL of the predecessor is copied. All the coincident edges' polygons are then either removed from or added to the NL according to the previously mentioned criteria. The resulting NL is then copied to all the coincident edges as their NLs.

Now we are going to examine the edges in the below edge list. There exist a few cases for an edge e:

1. SP corresponding to e is not an intersection

(a) e is visible and e's polygon is to its left

(b) e is visible and e's polygon is to its right

(c) e is invisible

2. SP corresponding to e is an intersection

(a) e was visible before the intersection and e's polygon is to its left

i. e is still visible

ii. e becomes invisible

(b) e was visible before the intersection and its polygon is to its right

i. e is still visible,

ii. e becomes invisible

(c) e was not visible before the intersection

i. e becomes visible and its polyon is to its left

ii. e becomes visible and its polygon is to its right

iii. e is still invisible

Each case has its own characteristics, and thus we need to handle them in different ways. We elaborate how to deal with the edges case by case.

B.2.1 SP Is Not An Intersection

If the SP related to edge e is not an intersection, the SP is either a top vertex or a side vertex, and e just starts at this SP. First, let us consider when e is visible and its polygon is to its left. We output a $2D$ vertex to e's polygon at the current SP. e's left visible segment is output to e's polygon. The shallowest polygon

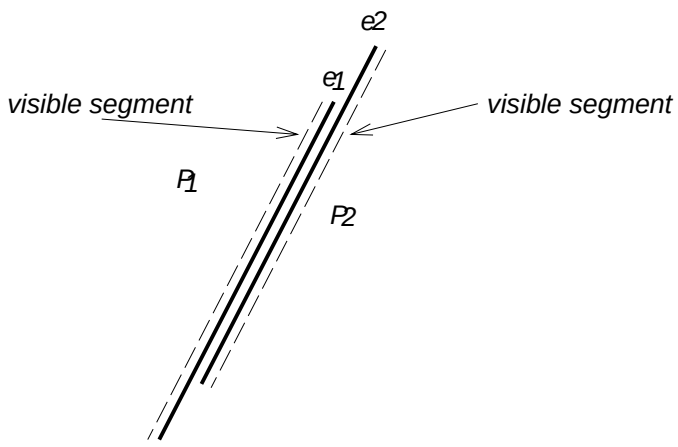


Figure B.4: Visible Segments and Coincident Edges

right of e should also have a $2D$ vertex at this SP and a visible segment along e. However, if that polygon is the polygon of e's coincident edge, we do not output anything for this polygon because we will deal with this visible segment when we handle that coincident edge (See Figure B.4). In the figure, edges e_1 and e_2 and their respective visible segments do not overlap. This is a distortion for a clear illustration, in reality they do overlap. The case that e is visible and its polygon is to its right is treated in a similar manner. We do not need to do anything in the case that e is invisible.

B.2.2 SP Is An Intersection

e was visible

If the SP corresponding to e is an intersection, e was visible before the intersection, and e's polygon is to its left, e might be still visible or become invisible after the intersection. In the case that e is still visible, if the polygon to e's right before the

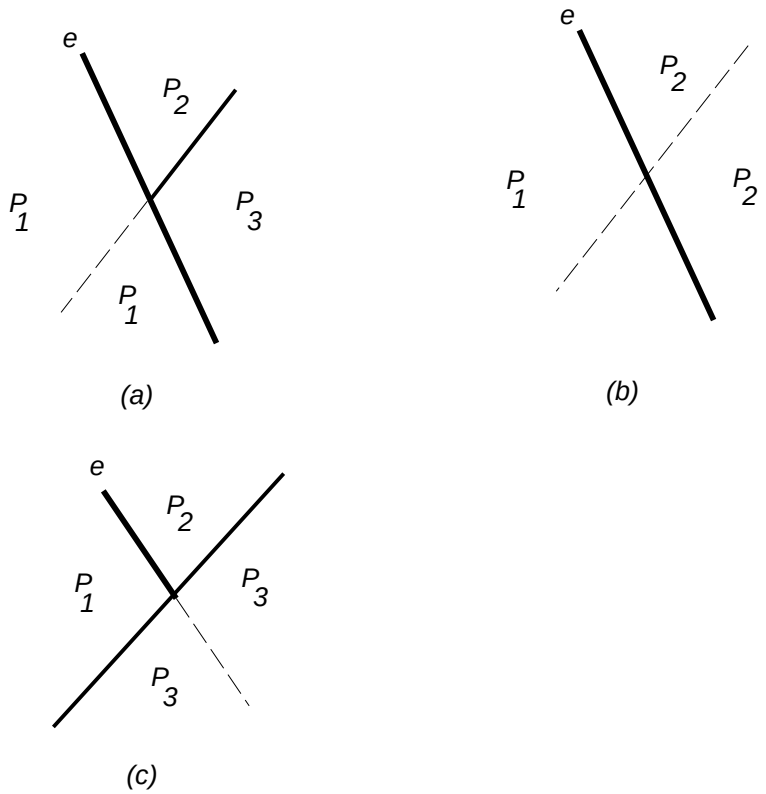


Figure B.5: Intersection Sweeping Points

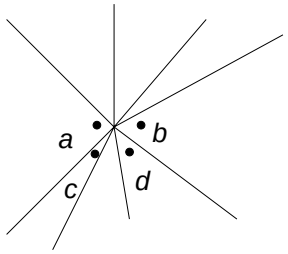


Figure B.6: 2 Variable Constraints Between Above And Below Vertices

intersection is not the same one after the intersection as shown in Figure B.5(a), we output a $2D$ vertex to polygon P_2 at this SP, and close e's right visible segment by giving this vertex as its end vertex in case e has a right visible segment. Here a three-variable constraint, among P_2's $2D$ vertex at this SP and the start and end vertices of e's left visible segment, is output. The start vertex is not actually known yet and this three-variable constraint will be completed when it is known. We also output a $2D$ vertex to polygon P_3. A three-variable constraint is also output among P_3's $2D$ vertex at this SP and the start and end vertices of e's left visible segment. P_3 should have a visible segment along e as well. As in Case 1.(a), we output e's right visible segment for P_3 if P_3 is not the polygon of an e's coincident edge. Otherwise, if the polygon to e's right before the intersection is the same one as after the intersection as shown in Figure B.5(b), nothing needs to be done. In the case that e becomes invisible as illustrated in Figure B.5(c), we output a $2D$ vertex to e's polygon. e's left visible segment is closed by assigning this vertex as its start vertex. All the three-variable constraints related to e's left visible segment are completed with this start vertex. If e has a right visible segment, we output a $2D$ vertex to the polygon to e's right. e's right visible segment is closed with this vertex as its end vertex. If e was visible before the intersection and its polygon is to its right, e is treated in a similar way.

e was not visible

If the SP corresponding to e is an intersection, and it was not visible before the intersection, e may become visible or may still be invisible. If e becomes visible and its polygon is to its left, we output a $2D$ vertex to this polygon at current SP. e's left visible segment is output to this polygon with that vertex as its end vertex. If the shallowest polygon right of e is not the polygon of e's coincident edge, a $2D$ vertex is output to that polygon, and e's right visible segment is output to the shallowest polygon right of e with that vertex as its start vertex. Otherwise, nothing is done at this time because we deal with it when we handle that coincident edge. If e becomes visible and its polygon is to its right is a case that is dealt with in a similar approach. If e is still invisible, we do not need to do anything with it.

Similar to the treatment of edges in the above edge list, we create two-variable constraints among the $2D$ vertices output corresponding to those edges in the below edge list. One two-variable constraint is created between the first $2D$ vertex above and the first $2D$ vertex below, and another two-variable constraint is also created between the last $2D$ vertex above and the last $2D$ vertex below. For example, in Figure B.6, vertices a and b are created during the processing of above edges, and c and d are created during the processing of below edges. a and c have a two-variable constraint, so do b and d.

B.3 Running Time

Suppose that there are n vertices and k edge intersections in a scene, the sweeping procedure stops $n + k$ times in total. Because in a convex polygon the number of

edges is equal to the number of vertices, there are n polygon edges in this scene. We know that each sweeping point has two corresponding edges. To each edge, all the other operations can be done in constant time except searching for an edge in AEL, inserting an edge into the AEL, deleting an edge from AEL, copying a node list, inserting a polygon into a node list, deleting a polygon from a node list, inserting an intersection into the priority queue, and completing the three-variable constraints for an edge. AEL, node list, and priority are all implemented as linked lists. The running time of searching, inserting, and deleting to an AEL is proportional to the size of AEL, which in the worst case is n, the number of edges in the scene. The running time of copying a node list in worst case is proportional to the number of polygons in the scene, which is smaller than n. Inserting an intersection into the priority queue takes $O(n)$ time. The number of uncompleted three-variable constraints for an edge, in worst case, is also proportional to the number of edges in the scene. Therefore, the worst case running time for sweeping is $O((n + k)n)$.

Appendix C

Data Structures and Pseudo-Code

This chapter lists the C definitions of some major data structures used in the constraint generation and the C like pseudo-code for sweeping algorithm.

C.1 Data Structures

```
typedef double Real;
/* so we can switch between float or double easily */

/* the structure for a 3D vertex, the input vertex */
typedef struct VertexStr {
  Real x, y, z;            /* 3D coordinates */
  Real hx, hy, hz, hw;     /* homogeneous coordinates */
  Real r0, g0, b0;         /* initial r, g, b */
} Vertex;

/* the structure for a 2D vertex, the output vertex */
typedef struct Vertex2DStr {
  Real x, y;   /* the 2D coordinates */
  int id;   /* the id number of the vertex used for optimization */
  Real r0, g0, b0;     /* initial r, g, b */
```

```
  Real r1, g1, b1;
  /* r, g, b after the first linear transformation
     r1, g1, and b1 also store h, s, v in HSV model after
     r1, g1, and b1 are transformed into h, s, and v, and other RGB
     values during other stages of the rendering pipeline */
  struct Vertex2DStr *next;
  /* the next output vertex in the polygon */
} Vertex2D, *Vertex2DPtr;

/* the first type strong constraint, V1 / V2 = C0 */
typedef struct EdgeCons1Str {
  Vertex2D *v1, *v2;      /* the first and second vertex */
  Real C0;                /* the initial contrast */
} EdgeCons1;

/* C0 = V1 / (t*V2 + (1-t)*V3) when the Tag is EDGE_CONS2;
   C0 = (t*V2 + (1-t)*V3) / V1 when Tag is EDGE_CONS3 */
typedef struct EdgeCons2Str {
  Vertex2D *v1, *v2, *v3; /* the first, second, and third vertex */
  Real C0;                /* intial contrast */
  Real t;                 /* projection of v1 on segment v2, v3 */
} EdgeCons2_3;

/* an enumeration for edge constraint tag */
typedef enum {EDGE_CONS1, EDGE_CONS2, EDGE_CONS3} EdgeConTag;

/* type definition for strong constraints across edges */
typedef struct EdgeConsStr {
  EdgeConTag tag;
  union {
    EdgeCons1 *cons1;
    EdgeCons2_3 *cons2;
  } Con;
  struct EdgeConsStr *next;
  /* a pointer to next constraint in the list of constraints */
} EdgeCons;

/* the structure of visible segments */
typedef struct VisibleSegmentStr {
```

```
  Vertex2D *start, *end;        /* the indices of output vertices */
  struct VisibleSegmentStr *next;
  /* the next visible segment in that polygon */
  short thread;          /* 1 is a thread, 0 nope */
  struct MasterPolyStr *master;
  /* the master polygon, this visible segment belons to */
} VisibleSegment;

typedef enum {NONE, LEFT, RIGHT} LeftOrRight;

/* the uncompleted constraint list for an edge */
typedef  struct Constraint3Ptr {
    EdgeCons *EdgeC;
    struct Constraint3Ptr *next;
} CListForEdge;

/* the structure for an edge */
typedef struct EdgeStr {
  struct  MasterPolyStr *poly,
          /* pointer to the polygon, the edge belongs */
  struct  MasterPolyStr *LeftPoly, *RightPoly;
  /* the right and left topest polygons of an edge before a sweeping
     point these pointers are used to decide the cases when a SP is
     an intersection */
  Vertex *first, *second;
  /* the start and end vertices of an edge */
  VisibleSegment *left, *right;
  /* left and right visible segments */
  LeftOrRight PolyLocation;
  /* Polygon is in the left or right side of the edge */
  LeftOrRight  coin;
  /* whether this edge coidcides with other edges, and if
     so the type LEFT: this edge's polygon is to the left of
     this edge RIGHT: this edge's polygon is to the right of
     this edge */
  struct EdgeStr *shared;
  /* the coincident edge, shared means the edge shares the same
     position As stated in Chapter 5, we really concern the  LEFT
     and RIGHT coincident edges whose polygons are the frontest in
```

```
     the left and right respectively, so only one pointer is enough
     instead of a list of coincident edges */
  Vertex2D *StartVis;
  /* vertex where this edge starts to be visible */
  /* it is used to specify one of the starting/ending vertices of a
     visible edge for a 3 variable constraint */
  CListForEdge *CList;
  /* list of uncompleted 3 variable constraints */
} Edge;

/* the structure for polygon information */
typedef struct PolyInfoStr {
  Vertex *first;         /* the list of 3D vertices */
  int num;               /* number of 3D vertices */
  Real A, B, C, D;       /* the plane equation of the polygon */
  Vertex2D *OutFirst;
  /* the output vertices after visible surface determination */
  Vertex2D *OutLast;     /* pointer to the last such vertex */
  VisibleSegment *SegFirst;  /* the first visible segment */
  VisibleSegment *SegLast;   /* the last visible segment  */
} PolyInfo;

/* the structure for a hole */
typedef struct HoleInfoStr {
   Vertex2DPtr *first;   /* list of 2D vertices for the hole */
   int num;
} HoleInfoType;

/* the information for a slave polygon, a slave polygon is a new
   polygon which was part of a master polygon before the
   visible algorithm determination */
typedef struct SlavePolyInfoStr {
  Vertex2DPtr *first;    /* array of vertices for the out contour */
  int num;               /* number of vertices */
  HoleInfoType *holes;   /* the array of holes in the polygon */
  int HoleNum;           /* number of holes */
} SlavePolyInfo;

/* the structure of slave polygon */
```

```
typedef struct SlavePolyStr {
  SlavePolyInfo poly;
  struct MasterPolyStr *master;  /* the master polygon */
  struct SlavePolyStr *next;     /* next slave polygon */
} SlavePoly;

/* master polygon is a polygon before visible
   surface determination */
typedef struct MasterPolyStr {
  PolyInfo poly;         /* the information about this polygon */
  SlavePoly *slave;      /* the first slave polygon */
} MasterPoly, *MasterPolyPtr;

/* the structure of master polygon list */
typedef struct MasterPolyListStr {
  MasterPolyPtr *master;  /* array of pointers to master polygons */
  int n;                  /* number of master polygons */
} MasterPolyList;
```

C.2 Pseudo-Code of Sweeping Algorithm

```
/* sweep until the PQ is empty */
void SweepAll( ) {
   while (PQ is not empty) {
      S1 = DeleteSameSPs();
      /* delete all the SPs with the same XY at the top of
         PQ from PQ, S1 is a set */
      SweepMult(S1);
   } /* while */
} /* SweepAll */

/* output a 2D vertex to polygon Poly, if a 2D vertex has
   already exist at this SP, just retrun the existing one */
Vertex2D *OutputVertex(Poly, SP) {

  if(poly == NULL) return;
  GetSweepXY(SP, &X, &Y);
```

```
  if (Poly->poly.OutLast != NULL && Poly->poly.OutLast->x == X &&
      Poly->poly.OutLast->y == Y) return(Poly->poly.OutLast);

  OutVer = (Vertex2D*) malloc(sizeof(Vewrtex2D));
  OutVer->x = X;
  OutVer->y = Y;
  Compute the colour for OutVer;
  poly->OutLast->next = OutVer;
  OutVer->next = NULL;
  return(OutVer);

} /* OutputVertex */

/* process the edges corresponding to the SPs in S1 */
void SweepMult(S1) {

  /* sort all the corresponding edges above the current SPs
     and keep them in a Above Edge List */
  SortAndKeepAbove(S1);
  /* sort all the corresponding edges below the current SPs
     and keep them in a Below Edge List */
  SortAndKeepBelow(S1);

  /* handle all the edges above these sweeping points */
  e = FirstAboveEdge(&SP);
  /* get first edge in the above edge list,
     SP is the sweeping point corresponding to e */
  while (e != NULL) {
  /* there are edges in the above edge list */

    if (SP is an intersection) {
       e->LeftPoly = The shallowest polygon left of e;
       e->RightPoly = The shallowest polygon right of e;
       /* remember these information For handling e in the
          below edge list */

    } else { /* SP is not an intersection, e ends at SP */
       if (e->left != NULL) {
       /* e has left visible segment, e is visible */
```

```
         PrePoly = e->left->master;
         OutVer = OutputVertex(PrePoly, SP);
         /* output a 2D vertex to polygon left of e at SP */
         e->left->start = OutVer;
         /* close the left visible segment */
         e->left = NULL;
         if (e->PolyLocation == LEFT)
             CompleteConstraints(e, OutVer);
         /* complete the 3 variable constraints
            associated with e */
         if (SP is a bottom vertex) Create a thread;
      } /* if */

      if (e->right != NULL) {
         SuccPoly = e->right->master;
         OutVer = OutputVertex(OutPoly, SP);
         /* output a 2D vertex to polygon right of e at SP */
         e->right->end = OutVer;
         /* close the right visible segment */
         e->right = NULL;
         if (e->PolyLocation == RIGHT)
            CompleteConstraints(e, OutVer);
         /* complete the 3 variable constraints
            associated with e */
      } /* if */

   } /* else */

   /* get the next edge in the above edge list */
   e = NextAboveEdge(&SP);
} /* while */

Create 2 variable constraints among the 2D vertices
   output For edges above SP;
Delete all the edges in the Above Edge List from AEL;

/* Insert all the edges in the Below Edge List into AEL */
e = FirstBelowEdge(&SP);
/* get the first edge in the below edge list */
```

```
while (e != NULL) {
  Insert e into AEL;
  NodeL = The Node List of the edge before e in the AEL;
  e1 = NextBelowEdge(&SP);
  CoinList = {};
  /* the coincident edge list is empty at begin */

  while  (e1 is coincident to e) {
    CoinList += {e1};  /* insert e1 to coincident edge */
    if (e1->PolyLocation == LEFT)
       /* e's polygon is to e's left */
       Delete e1->poly from NodeL;
    else /* e's polygon is to e's right */
       Insert e1->poly into NodeL;
    e1 =  NextBelowEdge(&SP);
  } /* while */

  e's NL is copied from NodeL;
  Insert all the edges in CoinList into AEL and their NLs
  are all copied from NodeL;

  e = e1;
} /* while */

/* handle all the edges below the SP */
e = FirstBelow(&SP);
/* the first edge in the below edge list */
while (e != NULL) {
/* there are edges left in the below edge list */
  PrePoly = the shallowest polygon to the left of e;
  SuccPoly = the shallowest polygon to the right of e;
  if (SP is not an intersection) {
     if (PrePoly == e->poly) {
     /* e is visible, its poly is to its left */
        OutVer = OutputVertex(PrePoly, SP);
        e->StartVis = OutVer; /* where e starts to be visible */
        e->left = OutputVisibleSegment(PrePoly, NULL, OutVer);
        /* OutVer is the ending vertex of this visible
           segment, because we have not known the starting
```

```
            vertex yet, use NULL.  this visible segment will be
            completed later by assigning a starting
            vertex For it */

          if (SuccPoly is not the polygon of e's coincident edge) {
            OutVer = OutputVertex(SuccPoly, SP);
            e->right = OutputVisibleSegment(SuccPoly, OutVer, NULL);
            /* we have not known the ending vertex yet */
          } else e->right = NULL;

       } else if (SuccPoly == e->poly) {
         /* e is visible,its poly is to its right */

         if (PrePoly is not the polygon of e's coincident edge) {
            OutVer = OutputVertex(PrePoly, SP);
            e->left = OutputVisibleSegment(PrePoly, NULL, OutVer);
         } else e->left = NULL;

         OutVer = OutputVertex(SuccPoly, SP);
         e->StartVis = OutVer;
         e->right = OutputVisibleSegment(SuccPoly, OutVer, NULL);

       } else {  /* e is invisible */
         e->left = NULL;
         e->right = NULL;
       } /* else */

    } else { /* SP is an intersection */

      if (e->LeftPoly == e->poly) {
      /* e was visible before the intersection,
         its poly is/was to its left */

         if (PrePoly == e->poly) { /* e is still visible */

           if (e->RightPoly != SuccPoly) {
           /* 3 variable constraints needed */
              if (e->RightPoly != NULL) {
                 OutVer = OutputVertex(e->RightPoly, SP);
```

```
            if (e->right != NULL) {
               e->right->end = OutVer;
               e->right = NULL;
            } /* if */
            CreateConstraint3(e, OutVer, e->StartVis);
            /* output a 3 variable constraint associated
               with e.  we only know two variables right now.
               this constraint will be completed when another
               vertex of e's left visible segment is known */
         } /* if */

         if (SuccPoly ! = NULL) {
           OutVer = OutputVertex(SuccPoly, SP);
           if (SuccPoly is not the polygon of
               e's coincident edge)
              e->right =
               OutputVisibleSegment(SuccPoly, OutVer, NULL);
           CreateConstraint3(e, OutVer, e->StartVis);
         } /* if */

      } /* if */

   } else { /* e becomes invisible */

      OutVer = OutputVertex(e->LeftPoly, SP);
      e->left->start = OutVer;
      CompleteConstraints(e, OutVer);
      /* complete the 3 variable constraints by giving
         the starting vertex of e's left visible segment */

      if (e->right != NULL) {
         OutVer = OutputVertex(e->RightPoly, SP);
         e->right->end = OutVer;
      } /* if */

      e->left = NULL:
      e->right = NULL;
   } /* else */
```

```
} else if (e->RightPoly == e->poly) {
  /* e was visible before intersection*/

   if (SuccPoly == e->poly) { /* e is still visible */
      if (e->LeftPoly != PrePoly) {
      /* 3 variable constraint */
         if (e->LeftPoly != NULL) {
            OutVer = OutputVertex(e->LeftPoly, SP);
            if (e->left != NULL) {
               e->left->start = OutVer;
               e->left = NULL;
            } /* if */
            CreateConstraint3(e, OutVer, e->StartVis);
         } /* if */

         if (PrePoly != NULL) {
            OutVer = OutputVertex(PrePoly, SP);
            if (PrePoly is not the polygon of
                e's coincident edge) e->left =
                OutputVisibleSegment(PrePoly, NULL, OutVer);
            CreateConstraint3(e, OutVer, e->StartVis);
         } /* if */
      } /* if */

   } else { /* e becomes invisible */
      OutVer = OutputVertex(e->RightPoly, SP);
      e->right->end = OutVer;
      CompleteConstraints(e, OutVer);

      if (e->left != NULL) {
         OutVer = OutputVertex(e->LeftPoly, SP);
         e->left->start = OutVer;
      } /* if */

      e->left = NULL;
      e->right = NULL;

   } /* else */
```

```
      } else { /* e was invisible before the intersection */

         if (PrePoly == e->poly) {
         /* e becomes visible and its poly is to its left */
            OutVer = OutputVertex(PrePoly, SP);
            e->left = OutputVisibleSegment(PrePoly, NULL, OutVer);
            e->StartVis = OutVer;

            if (SuccPoly is not the polygon of e's
                coincident edge) {
               OutVer = OutputVertex(SuccPoly, OutVer);
               e->right = OutputVisibleSegment(SuccPoly,
                                        OutVer, NULL);
            } /* if */

         } else if (SuccPoly == e->poly) {
         /* e becomes visible and its poly is to its left */
           OutVer = OutputVertex(SuccPoly, SP);
           e->right = OutputVisibleSegment(SuccPoly,
                                     OutVer, NULL);
           e->StartVis = OutVer;

           if (PrePoly is not the polygon of e's
               coincident edge) {
             OutVer = OutputVertex(PrePoly, SP);
             e->left = OutputVisibleSegment(PrePoly,
                                    NULL, OutVer);
           } /* if */

         } /* else if */

      } /* else, e was invisible */

   } /* else, SP is an intersection */

   Create 2 variable constraints among the vertices
      output with the edges below the SP;
   Create a 2 varibele constraint between the first
      above 2D vertex and the first below vertex;
```

```
    Create a 2 variable constraint between the last
       above 2D vertex and the last below vertex;
    /* it is possible that sometimes the first and the last
       above vertices are the same one, so are the below
       vertices. In that case, caution should be spent.
       The principle is that the neighbouring 2D vertices
       whose visible segments have the relationship as described
       in the 2 variable constraint case should have
       a 2 variable constraint */

  } /* while */

} /* SweepMult */
```

Bibliography

[AK87] J. Arvo and D. Kirk. Fast ray tracing by ray classification. In *SIGGRAPH '87 Proceedings*, pages 55–64, 1987.

[Ama84] J. Amanatides. Ray tracing with cones. In *SIGGRAPH '84 Proceedings*, pages 129–135, 1984.

[BKT86] K.R. Boff, L. Kaufman, and J.P. Thomas, editors. *Handbook of Perception and Human Performance*, volume I. John Wiley & Sons, Now York, 1986.

[Bli92] J.F. Blinn. Hyperbolic interpolation. *IEEE Computer Graphics and Applications*, 12(4):89–94, 1992.

[BN78] J.F. Blinn and M.E. Newell. Clipping using homogeneous coordinates. In *SIGGRAPH '78 Proceedings*, pages 245–251, 1978.

[BT75] P. Bui-Tuong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[CG85] M.F. Cohen and D.P. Greenberg. The hemi-cube: A radiosity solution for complex environments. In *SIGGRAPH '85 Proceedings*, pages 31–40, 1985.

[CGIB86] M.F. Cohen, D.P. Greenberg, D.S. Immel, and P.J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(3):26–35, March 1986.

[CHS+93] K. Chiu, M. Herf, P. Shirley, S. Swamy, C. Wang, and K. Zimmerman. Spatially nonuniform scaling functions for high contrast images. In *Proceedings of Graphics Interface '93*, pages 245–253, 1993.

[Coo86] R.L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[Cor70] T.N. Cornsweet. *Visual Perception*. Academic Press, 1970.

[Cow87] W.B. Cowan. Colorimetric properties of video monitors. Notes for a short course presented at the Annual Meeting of the Optical Society of America, October 1987.

[Cow93] W.B. Cowan. Personal communication, 1993.

[CPC84] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH '84 Proceedings*, pages 137–145, 1984.

[CSW87] W.B. Cowan, M. Stone, and C. Ware. Colour perception. Graphics Interface '87 and SIGCHI '87 Tutorial Notes, 1987.

[FLS77] R.P. Feynman, R.B. Leighton, and M. Sands. *The Feynman Lectures on Physics*, volume 1, chapter 35. Addison-Wesley Publishing Company, 1977. also in J.C. Beatty and K.S. Booth (eds.) Tutorial: Computer Graphics, pp. 354-363.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, second edition, 1990.

[GFMS93] A.S. Glassner, K.P. Fishkin, D.H. Marimont, and M.C. Stone. Rendering within constraints. Submitted to ACM Transactions on Graphics, 1993.

[GHMS86] P.E. Gill, S.J. Hammarling, W. Murray, and M.A. Saunders. *User's Guide for LSSOL (Vertion 1.0): A Fortran package for constrained linear least-squares and convex quadratic programming*. Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305, 1986.

[Gim87] P. Gimferrer. *Magritte.* Academy Editions, London, 1987.

[Gla89] A.S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, San Diego, 1989.

[Gla90] A.S. Glassner, editor. *Graphics Gems*. Academic Press, San Diego, 1990.

[GMSW84] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright. Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Math. Software,* 10:282–298, 1984.

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Trans. on Computers,* C-20(6):623–629, June 1971.

[GTGB84] C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84 Proceedings*, pages 213–222, 1984.

[Hal86] R. Hall. Hybrid techniques for rapid image synthesis. In T. Whitted and R. Cook, editors, *Image Rendering Tricks*. Course Notes 16 for SIGGRAPH 86, Dallas, 1986.

[HG77] G Hamlin and C.W. Gear. Raster-scan hidden surface algorithm techniques. In *SIGGRAPH '77 Proceedings*, pages 206–213, 1977.

[HG83] R.A. Hall and D.P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, November 1983.

[HG86] E.A. Haines and D.P. Greenberg. The light buffer: A shadow-testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, September 1986.

[HH84] P.S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH '84 Proceedings*, pages 119–127, 1984.

[ICG86] D.S. Immel, M.F. Cohen, and D.P. Greenberg. A radiosity method for non-diffuse environments. In *SIGGRAPH '86 Proceedings*, pages 133–142, 1986.

[JW75] D.B. Judd and G. Wyszecki. *Color in Business, Science and Industry*. John Wiley & Sons, New York, 3rd edition, 1975.

[Kaj86] J.T. Kajiya. The rendering equation. In *SIGGRAPH '86 Proceedings*, pages 143–150, 1986.

[Kla89] R. V. Klassen. *Device Dependent Image Construction for Computer Graphics*. PhD thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1989.

[KR88] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1988.

[Mac91] B. MacIntyre. A constraint-based approach to dynamic colour management for windowing interfaces. Master's thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1991.

[MC92] B. MacIntyre and W. B. Cowan. A practical approach to calculating luminance contrast on a CRT. *ACM Transactions on Graphics*, 11(4):336–347, Oct. 1992.

[Meh84] K Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Monographs on Theoretical Computer Science (Editors: W. Brauer, G. Rozenberg, and A. Salomaa). Springer-Verlag, Berlin, 1984.

[NAG91] NAG. *NAG Fortran Library Manual Mark 15*. Numerical Algorithms Group Limited, 1991.

[Nur85] O. Nurmi. A fast line-sweep algorithm for hidden line elimination. *BIT*, 25(3):466–472, 1985.

[Nye88] Adrian Nye. *Xlib Programming Manual*, volume One of *X Window System*. O'Reilly & Associates, Inc, 1988.

[Nye90] Adrian Nye, editor. *Xlib Reference Manual*, volume Two of *X Window System*. O'Reilly & Associates, Inc, second edition, 1990.

[Rat72] F. Ratliff. Contour and contrast. *Scientific American*, pages 90–101, 1972. also in J.C. Beatty and K.S. Booth (eds.) Tutorial: Computer Graphics, pp. 365-375.

[SCB87] M.W. Schwarz, W.B. Cowan, and J.C. Beatty. An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models. *ACM Transactions on Graphics*, 6(2):123–158, 1987.

[Sch90] K.G. Schlueter. Perceptual synchronization in window systems. Master's thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, 1990.

[SG82] S. Sechrest and D.P. Greenberg. A visible polygon reconstruction algorithm. *ACM Transactions on Graphics*, 1(1):25–42, 1982.

[SH76] M.I. Shamos and D. Hoey. Geometric intersection problems. In *17th IEEE Symposium on Foundations of Computer Science*, pages 208–215, 1976.

[Smi78] A.R. Smith. Color gamut transform pairs. *Computer Graphics*, 12(3):12–19, 1978.

[Sob65] J.T. Soby. *René Magritte*. The Museum of Modern Art, New York, 1965.

[SS60] S.S. Stevens and J.C. Stevens. Brightness function: Parametric effects of adaptation and contrast. *Journal of the Optical Society of America*, 50(11):1139, November 1960.

[SS63] J.C. Stevens and S.S. Stevens. Brightness function: Effects of adaptation. *Journal of the Optical Society of America*, 53(3):375–385, March 1963.

[STN87] M. Shinya, T. Takahashi, and S. Naito. Principles and applications of pencil tracing. In *SIGGRAPH '87 Proceedings*, pages 45–54, 1987.

[Sto71] J. Stoer. On the numerical solution of constrained least-squares problems. *SIAM J. Num. Anal.*, 8:382–411, 1971.

[TR91] J. Tumblin and H. Rushmeier. Tone reproduction for realistic computer generated images. Technical Report GIT-GVU-91-13, Graphics, Visualization, & Usability Center, College of Computing, Georgia Institute of Technology, Atlanta GA, 1991.

[WCG87] J.R. Wallace, M.F. Cohen, and D.P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *SIGGRAPH '87 Proceedings*, pages 311–320, 1987.

[WHG84] H. Weghorst, G. Hooper, and D.P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.