



Support for the camera model
by Gabor Kincses

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science

Montana State University

© Copyright by Gabor Kincses (1992)

Abstract:

In three-dimensional photorealistic rendering, one of the most commonly used projection models is the camera model. Unfortunately, its complexity often makes it difficult to generate satisfactory pictures. Therefore we developed a system which allows the user to specify where or how the displaying objects should appear on the target screen. From this information, our program calculates the best possible position of the imaginary camera. This position can be supplied to any of the existing software/hardware camera transformations, which then displays the desired pictures. This system is an effective method to help the graphics programmer generate three-dimensional images.

**SUPPORT FOR THE
CAMERA MODEL**

by

Gabor Kincses

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 1992

N378
K5743

-ii-

APPROVAL

of this thesis submitted by

Gabor Kincses

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style and consistency, and is ready for submission to the College of Graduate Studies.

May 29, 1992
Date

J. Dunbig Starch
Chairperson, Graduate Committee

Approved for the Major Department

May 29, 1992
Date

J. Dunbig Starch
Head, Major Department

Approved for the College of Graduate Studies

May 29, 1992
Date

Henry Parsons
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Dean of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature

Gale K. K.

Date

5/29/92

TABLE OF CONTENTS

LIST OF FIGURES	v
ABSTRACT	vi
1. THE BASICS OF THE CAMERA MODEL	1
Overview of Projections	1
A Special Case	3
The General Case	4
Solution for the General Case	5
2. A "RULE-BASED" SYSTEM	8
The Problem	8
Subrule Prototypes	9
The First Subrule	9
The Second Subrule	11
The Loose Interpretation	11
3. SOLVING THE EQUATION SYSTEM	13
Generating Equations	13
Three Locations	14
Two Locations, One Orientation	15
One Location, Two Orientations	15
Three Orientations	17
The Non-Linear Equation System Solver	17
Singularities	18
4. CONCLUSIONS	19
5. FUTURE ENHANCEMENTS	20
REFERENCES	21

LIST OF FIGURES

1. Perspective Projection	3
2. Transformation from World Space to View Space	6
3. Perspective Projection in View Space	10

ABSTRACT

In three-dimensional photorealistic rendering, one of the most commonly used projection models is the camera model. Unfortunately, its complexity often makes it difficult to generate satisfactory pictures. Therefore we developed a system which allows the user to specify where or how the displaying objects should appear on the target screen. From this information, our program calculates the best possible position of the imaginary camera. This position can be supplied to any of the existing software/hardware camera transformations, which then displays the desired pictures. This system is an effective method to help the graphics programmer generate three-dimensional images.

CHAPTER 1

THE BASICS OF THE CAMERA MODEL

Overview of Projections

Comparing the two-dimensional viewing process to the three-dimensional, one can readily realize that the latter is significantly more complex. For example, in two dimensions the *view port* acts as a simple window which can be moved around in the two-dimensional world. This means that any given point in the two-dimensional world coordinate system transforms into an image point in view port coordinates by a simple translation.

In three dimensions, however, the view port acts differently. Instead of cutting out a flat slice of the three-dimensional world as in two dimensions, the view port displays a projected two-dimensional image of the objects located in a three-dimensional space. Simulating the projection of the three-dimensional space onto the two-dimensional *view plane* is the heart of three-dimensional visualization. There are a number of projections available:

- Parallel projections

- Oblique
 - * Cabinet (30° or 45°)
 - * Cavalier (30° or 45°)
- Orthographic
 - * Top, Front, Side view
 - * Isometric
- o Perspective projections
 - One-point perspective
 - Two-point perspective
 - Three-point perspective

In three-dimensional visualization, the most commonly used viewing model is the three-point perspective projection; or in fashionable terms, the *camera model*. With a real-life camera, people generally take pictures and in doing so they perform a three-point perspective projection of the outside world (*world space*) onto a two-dimensional celluloid surface (*view plane*).

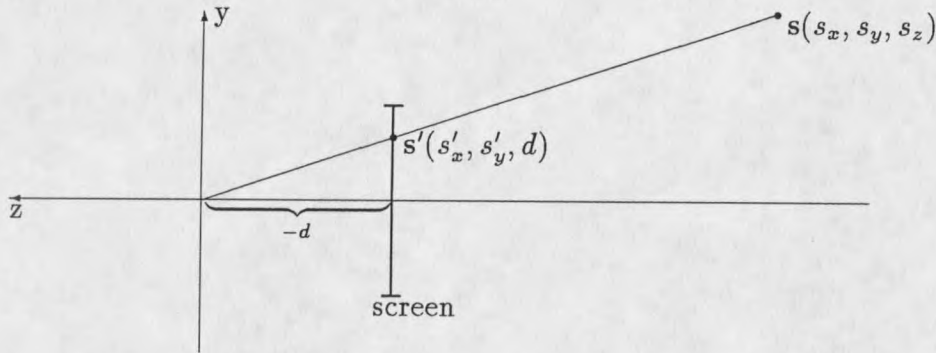


Figure 1: Perspective Projection

A Special Case

Most of the projections can be easily described by linear mathematical models. The camera transformation turns out to be rather complex, however, unless one uses a “nice” setting of the camera in the world coordinate system. For instance, if the camera is located at the origin of the coordinate system and it is looking in the direction of the z-axis and the *screen* is d units away with its center sitting on the z-axis (See Figure 1), then the transformation can be described by a (4x4) matrix using homogeneous coordinates:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \quad (1)$$

assuming column vectors. The simple calculation that results in this matrix can be found in [1] (Chapter 6.4.).

Unfortunately, this is a very special case and it is easy to realize that the position of the camera can be placed arbitrarily. Its position has ten degrees of freedom:

- *location* (3)
- *reference direction* (3)
- *up direction* (3)
- *field of view* (1).

The position of the screen is given by d . There is a simple relationship between the field of view (ϕ) and d :

$$d = -w/2 \cot(\phi/2) \quad (2)$$

where w is the width of the screen (See Figure 2).

The General Case

The reference direction identifies the points at which the camera is looking. After the camera's location and reference direction are fixed, it can still pivot around the reference direction. Once the up direction is chosen, the camera can no longer pivot around its axis, thus its position has been fully specified. The parameters camera location, reference direction and up direction can be represented by the vectors **cam**, **ref** and **up**, respectively. The field of view is historically denoted by ϕ .

In the above mentioned special case, these vectors have the following values:

$$\mathbf{cam} = (0, 0, 0) \quad (3)$$

$$\mathbf{ref} = (0, 0, -1) \quad (4)$$

$$\mathbf{up} = (0, 1, 0) \quad (5)$$

Note that $ref_z = -1$ means that a right-handed coordinate system is used.

Solution for the General Case

Performing the perspective transformation for the general case—which is an affine transformation from three dimensions to two dimensions with fixpoint **cam**—is rather complicated algebraically. There is a better way to find the projected point on the screen.

Given the transformation matrix for the perspective projection in the special case, it is obvious that if the general case can be reduced to the special case, the problem is solved. This reduction can easily be performed by transforming the original **cam**, **ref** and **up** vectors into something similar to the vectors given in (3), (4) and (5), respectively.

The assumption that all three vectors have unit lengths is not much restriction. In the general cases however, the **ref** and **up** vectors are not necessarily perpendicular

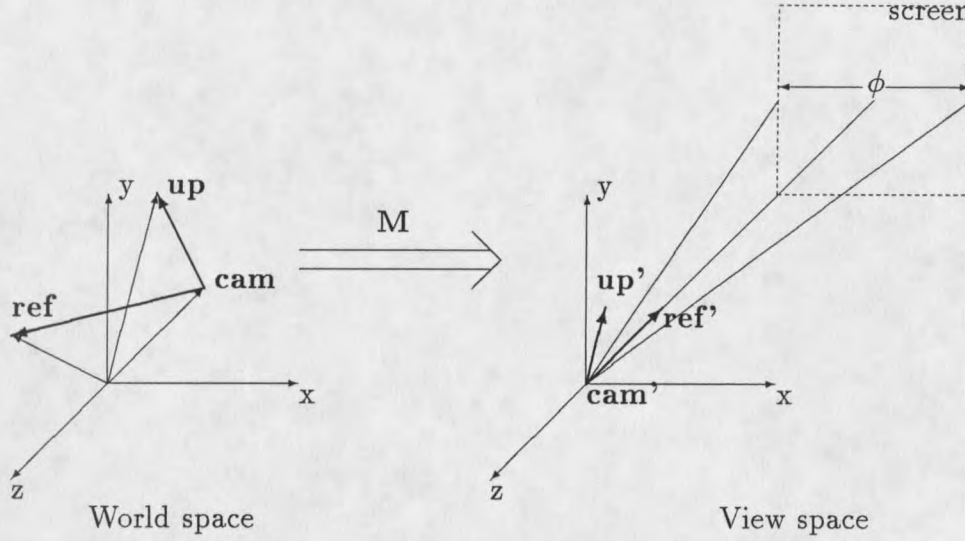


Figure 2: Transformation from World Space to View Space

to each other. This means that the transformed $\mathbf{up}'$ vector has the following form:

$$\mathbf{up}' = (0, up'_y, up'_z) \quad (6)$$

In other words, only its projection to the screen will be truly vertical.

The transformation from *world space* (S_1) to *view space* (S_2) is linear and isometric. It can be constructed from a single translation and a single rotation. The translation is very simple to define. It takes the position of the camera in S_1 to the origin of S_2 . This means that the translation is $-\mathbf{cam}$. The rotation can be calculated based on the above described facts:

- $\mathbf{ref}$ transforms into (4)
- $\mathbf{up}$ transforms into (6)

The transformation matrix using homogeneous coordinates and column vectors:

$$M = \begin{bmatrix} r_1 & r_2 & r_3 & 0 \\ up_x & up_y & up_z & 0 \\ -ref_x & -ref_y & -ref_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -cam_x \\ 0 & 1 & 0 & -cam_y \\ 0 & 0 & 1 & -cam_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

or after performing the multiplication:

$$M = \begin{bmatrix} r_1 & r_2 & r_3 & -cam_x r_1 - cam_y r_2 - cam_z r_3 \\ up_x & up_y & up_z & -cam_x up_x - cam_y up_y - cam_z up_z \\ -ref_x & -ref_y & -ref_z & cam_x ref_x + cam_y ref_y + cam_z ref_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

where

$$\begin{aligned} r_1 &= up_y ref_z - up_z ref_y \\ r_2 &= up_z ref_x - up_x ref_z \\ r_3 &= up_x ref_y - up_y ref_x \end{aligned} \quad (9)$$

The calculation can be found in [2].

CHAPTER 2

A "RULE-BASED" SYSTEM

The Problem

Although the above described ten adjustable parameters do not comprise a large system, it is large enough for the average graphics programmer to be often intractable, or at least confusing.

Typically the programmer defines a set of surface elements, which, in turn, form a number of objects. The objects are placed in world space in some fashion. The programmer has definite knowledge about the position of these objects, as well as about the picture he/she would like to see. Unfortunately, finding the best compromise for the camera position based on the conditions imposed by the user is not trivial. Calculating the camera position is usually a complex process, therefore most programmers simply approximate the desired result. To ease this process, a program was developed by the author which allows the user/programmer to specify a set of "desires" for which the best possible answer is calculated.

The system described in [2] offers a solution to this problem, but for a few restricted cases only. Because of its limitations, it addresses only a very few of the needs of

users. Here are a few of the problems with this system:

- the number of objects included in the constraints is only two
- the constraints are very restricted in order to have a closed solution
- in each case the user has to provide some information about the camera

The system described in this paper does not require much information about the camera, only the field of view, and a distance in the last discussed case. Instead of providing the up vector, the user can specify some view space information, like orientation or location in view space.

Subrule Prototypes

First, one can choose a simple set of subrules. The subrules deal with the *location* and the *orientation* of the objects in the scene. They simply define a relationship between the original and the transformed object. The location and the orientation of an object is given in the simplest possible way. The center of the object is considered to be the location, the direction of the primary axis to be the orientation of the object.

The First Subrule

The first subrule is based on the location of the center of an object (p). The assumed transformation associates a point (v) in view space with the point p . Since

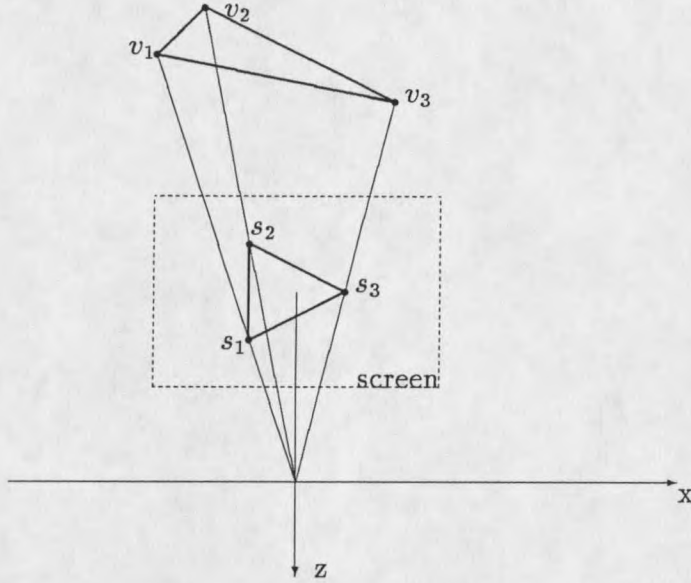


Figure 3: Perspective Projection in View Space

the user contemplates in screen coordinates rather than in view space coordinates, the user's choice defines a line instead of a certain point in view space. The line is easy to determine since it crosses the origin and the user specified screen location $s = [s_x, s_y, -w/2 \cot(\phi/2), 1]^T$. Thus the line can be given as:

$$\mathbf{v}(t) = \begin{bmatrix} t s_x \\ t s_y \\ -t w/2 \cot(\phi/2) \\ 1 \end{bmatrix} \quad (10)$$

This subrule can be given in the following closed form:

$$\mathbf{M} \mathbf{p} - \mathbf{v}(t) = 0 \quad (11)$$

The problem is the parameter t . To specify this parameter, the program in [2] uses the additional information of distance. The author chose a different way.

The Second Subrule

The second subrule deals with the orientation of the object. The user is allowed to specify the desired orientation of an object on the screen. Two different approaches can be taken.

First, we can assume that the orientation on the screen is the same as in view space. This means that the original orientation vector (w_o) transforms into the vector (s_o)—when applying the rotation of the transformation only—which is parallel to the screen. Thus $s_{oz} = 0$.

The other possibility is to assume that the chosen orientation is the projection of the real orientation. In this case the orientation lies in a plane which fits the z-axis and the user-selected orientation vector.

There can be a maximum of two orientations selected.

The equation generated by this rule can be written as:

$$Rw_o - s_o = 0 \quad (12)$$

where R comes from $M = TR$.

The Loose Interpretation

In this case the desired orientation vector lies in the plane created by the z-axis and the user-defined vector s_o . The equation of the plane is:

$$\frac{s_y}{s_x} x - y = 0 \quad (13)$$

The orientation vector is:

$$\mathbf{v}_o = \begin{bmatrix} t & s_x \\ t & s_y \\ z \\ 1 \end{bmatrix} \quad (14)$$

The parameters t and z can be determined from the constraints between the subrules.

The equation generated by this rule can be written as:

$$\mathbf{R}\mathbf{w}_o - \mathbf{v}_o(z, t) = 0 \quad (15)$$

CHAPTER 3

SOLVING THE EQUATION SYSTEM

Generating Equations

The equation system is a certain combination of the equations (11), (12) and (15).

It can be written as:

$$f(\bar{x}) = 0 \quad (16)$$

where $\bar{x}$ is the correct solution.

The equation (11) transforms into the following set of equations:

$$m_{11}p_x + m_{12}p_y + m_{13}p_z + m_{14} - v_x = 0 \quad (17)$$

$$m_{21}p_x + m_{22}p_y + m_{23}p_z + m_{24} - v_y = 0 \quad (18)$$

$$m_{31}p_x + m_{32}p_y + m_{33}p_z + m_{34} - v_z = 0 \quad (19)$$

The equations (12) and (15) transform into the following set of equations:

$$m_{11}w_{ox} + m_{12}w_{oy} + m_{13}w_{oz} - v_{ox} = 0 \quad (20)$$

$$m_{21}w_{ox} + m_{22}w_{oy} + m_{23}w_{oz} - v_{oy} = 0 \quad (21)$$

$$m_{31}w_{ox} + m_{32}w_{oy} + m_{33}w_{oz} - v_{oz} = 0 \quad (22)$$

As we saw above, the parameter t is undefined when the user selects a subrule dealing with location. In order to determine the value of t , one needs to obtain some further information in certain cases.

Three Locations

In the program described in this paper, the user can specify the location of up to three objects on the screen. Since the transformation M is isometric, the distance is preserved. This means that the distance between two vectors in world space must be equal to the distance between the two corresponding transformed vectors in view space:

$$||\mathbf{p}_2 - \mathbf{p}_1||^2 = ||\mathbf{v}_2 - \mathbf{v}_1||^2 \quad (23)$$

In the case of three locations, there can be three equations like (23) written, which is sufficient, since there are only three unknowns. Note that using squared-norms instead of simple norms saves the square-root operation. (23) is equivalent to the following:

$$\begin{aligned} & t_i^2 (s_{ix}^2 + s_{iy}^2 + (-w/2 \cot(\phi/2))^2) + \\ & + t_j^2 (s_{jx}^2 + s_{jy}^2 + (-w/2 \cot(\phi/2))^2) - \\ & - 2 t_i t_j (s_{ix} s_{jx} + s_{iy} s_{jy} + (-w/2 \cot(\phi/2))^2) - d_{i,j}^2 = 0 \end{aligned} \quad (24)$$

where

$$d_{i,j} = ||\mathbf{p}_i - \mathbf{p}_j|| \quad (25)$$

The equation system constructed from equations like (24) can be solved by the non-linear equation solver as described below.

Two Locations, One Orientation

When the user specifies two locations and one orientation, the number of unknown parameters is only two. (24) can be written for the distance between the two locations. The other equation can be generated after the fact that the dot-product of two vectors is preserved through the transformation M , since M is isometric. Thus the following must hold for M :

$$\frac{\mathbf{v}_2(t_2) - \mathbf{v}_1(t_1)}{\|\mathbf{v}_2(t_2) - \mathbf{v}_1(t_1)\|} \cdot \mathbf{v}_o = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \cdot \mathbf{w}_o \quad (26)$$

Expressing the t dependencies:

$$t_2 (s_{2x}v_{ox} + s_{2y}v_{oy}) - t_1 (s_{1x}v_{ox} + s_{1y}v_{oy}) - \mathbf{w}_o \cdot (\mathbf{p}_2 - \mathbf{p}_1) = 0 \quad (27)$$

Equations (24) and (27) provide a fully determined equation system for t_1 and t_2 , which can be solved by the non-linear equation solver as described below.

One Location, Two Orientations

In this case there is no way to specify the value of t in (11). Therefore one can simply set it to an arbitrary value, or the user can choose a suitable value. Note that in this case the angle between the two orientation vectors is preserved; therefore, one cannot choose two arbitrary orientations. Because of this, the first orientation chosen

by the user is interpreted as a strict orientation, the second is as a loose orientation.

Geometrically this means the following:

- the second orientation lies in the plane which fits the user defined orientation and the z-axis
- the length of the second orientation must be equal to 1, since the distance is preserved and the given orientation in world space has unit length

Summarizing the above statements mathematically:

$$\mathbf{w}_1 \cdot \mathbf{w}_2 = \mathbf{v}_1 \cdot \mathbf{v}_2 \quad (28)$$

$$v_{2x}^2 + v_{2y}^2 + v_{2z}^2 = 1 \quad (29)$$

$$v_{2y} = t s_{2y} \quad (30)$$

$$v_{2z} = t s_{2z} \quad (31)$$

(28) can be written as:

$$v_{1x}(s_{2x}t) + v_{1y}(s_{2y}t) + v_{1z}(s_{2z}t) - \mathbf{w}_1 \cdot \mathbf{w}_2 = 0. \quad (32)$$

Note, that $v_{1z} = 0$. Therefore t can be expressed from (32):

$$t = \frac{\mathbf{w}_1 \cdot \mathbf{w}_2}{v_{1x}s_{2x} + v_{1y}s_{2y}} \quad (33)$$

Knowing t , v_{2z} can be calculated from (29), (30) and (31):

$$v_{2z} = \sqrt{1 - (s_{2x}t)^2 - (s_{2y}t)^2} \quad (34)$$

Three Orientations

In this case three orientations should be specified. Since the distance and angles are preserved, the third orientation is either redundant or contradictory. Therefore this case should be omitted.

The Non-Linear Equation System Solver

The non-linear equation system that provides the solution to the problem consists of the nine equations generated by the three chosen subrules. The 12 variables in the transformation matrix M are not independent, however. The interdependencies between the variables are described by (9). These three equations complete the equation system, which is then non-singular.

Solving this equation system can be done by the generalized Newton-method. The Newton-method in the one-dimensional case uses the slope of the observed function to approximate it. In n -dimensions the gradient is used instead. The gradient is described by the Jacobian matrix:

$$J(x) = f'(x) = \begin{bmatrix} \partial f_1(x)/\partial x_1 & \dots & \partial f_1(x)/\partial x_n \\ \partial f_2(x)/\partial x_1 & \dots & \partial f_2(x)/\partial x_n \\ \vdots & & \vdots \\ \partial f_n(x)/\partial x_1 & \dots & \partial f_n(x)/\partial x_n \end{bmatrix} \quad (35)$$

The linear approximation of the function f is:

$$dx = J^{-1}(x) f(x) \quad (36)$$

which is an n -dimensional linear equation system, that can be solved by Gaussian elimination. The solution to this is the displacement of the next guess:

$$x_{new} = x_{old} + dx \quad (37)$$

Singularities

(26) describes how the dot-products are preserved for the case of one orientation and two locations. In order to be able to satisfy this condition the following inequality must hold:

$$\frac{s_2 - s_1}{\|s_2 - s_1\|} \cdot s_o \leq \frac{p_2 - p_1}{\|p_2 - p_1\|} \cdot w_o \quad (38)$$

In other words: the angle between the line through the two desired screen locations and the desired orientation must be less than or equal to the corresponding angle in world space. Otherwise no solution can be obtained.

(28) describes a similar relationship between the relative position of the two orientations. It states that the relative positions of the two orientations are maintained through the transformation. To be able to satisfy this condition the following inequality must hold:

$$w_{o1} \cdot w_{o2} \leq s_{o1} \cdot s_{o2} \quad (39)$$

CHAPTER 4

CONCLUSIONS

The goal of this system is to help the graphics programmer generate satisfactory three-dimensional images. This system is somewhat more involved than the one described in [2] and provides solutions for situations which are more likely to occur. The subrules dealing with locations turned out to be more useful than the ones which deal with orientations, although the orientation subrules are quite useful, too.

The system currently runs under X-windows, but it would not be too difficult to make it system independent. The reason for X-windows was only testing purposes.

The actual run-time of the calculation was not more than 0.5 seconds. This makes the program more convenient to use.

User-friendliness was an important issue in designing the system. Testing results show that it allows users to specify the constraints quite easily.

Generality could have been improved some. One can say that there are essentially three cases the program deals with. But one should also mention how powerful these cases are.

CHAPTER 5

FUTURE ENHANCEMENTS

The Newton-method does not have a convergence criterion. Therefore the so-called Line Search method needs to be implemented for robust operation. Throughout careful testing however, the iteration always converged.

One possible different approach could be to allow the user to specify an overdetermined system. In this case the system would generate all possible three-member combinations of the set of equations and would solve all of these. The solution with the least squared error would be the final solution.

Another possible approach is to use a neural network to find the optimum solution (best fit). The Hopfield network looks the most promising to do this.

A completely different approach could be an expert system, which would use set priorities between the subrules. The subrules should be somewhat more abstract more approaching the user's way of thinking.

REFERENCES

- [1] Foley, J.D., vanDam, A., Feiner, S.K., Hughes, J.F.,

Computer Graphics: Principles and Practice

Addison-Wesley Publishing Co., 1990.

- [2] Blinn, J., *Where am I? What am I looking at?*

IEEE Computer Graphics & Applications, Vol. 8, No. 7,

July 1988.

- [3] Maron, J.M., *Numerical Analysis*

McMillan Publishing Co., 1982.

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10179135 6

HOUCHEN
BINDERY LTD
UTICA/OMAHA
NE.