1986

Interactive Texturing For a Computer Graphics Display System

Yin L. Wah
University of Central Florida

INTERACTIVE TEXTURING
FOR A COMPUTER GRAPHICS DISPLAY SYSTEM

BY

YIN L. WAH
B.A., University of California, 1983

RESEARCH REPORT

Submitted in partial fufillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1986

ABSTRACT

Surfaces have texture, and few surfaces are truly smooth in the real world, yet this is how they are modeled. Texturing provides the illusion of increased surface roughness or granularity, and it also gives the illusion of extra detail and realism. This is accomplished by modulating surface color based on a gradient function known as texture pattern.

Changing texture pattern data can be expensive and time consuming since the change must be done on database offline system and then reloaded on the Computer Image Generator (CIG) for review of the change. The Interactive Texturing (IAT) system is designed for online examination and/or modification of the data associated with texture patterns on the General Electric Computer Image Generator (COMPUSCENE III CIG). With this online capability, the IAT system is very cost effective because it facilitates the process of debugging and improving new texture patterns.

In this paper, the texture pattern data, specifications of IAT system, interface requirements, the IAT system overview, limitations, measure of performance and future enhancement are described with respect to COMPUSCENE III CIG.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

CIG - Acronym for Computer Image Generator.

Compuscene I - A General Electric Computer Image Generator.

Compuscene II - A General Electric Computer Image Generator.

Compuscene III - A General Electric Computer Image Generator.

Compuscene IV - A General Electric Computer Image Generator.

IAT - Acronym for Interactive Texture.

LOD - Acronym for Level of Detail.

Level of Detail - Multiple versions for a single item. The different versions are based on distance of the item from the eye point within a data base, and the level of complexity for each version.

Online - This term refers to describe software that is executed with CIG.

Offline - This term refers to describe software that is executed without CIG.

Edge - A line segment formed between two vertices.

Face - A polygon with limitations as defined by the CIG it is to be displayed upon.

Polygon - A two-dimensional surface defined by vertices that are coplanar.

Surface - The exterior of an object or a plane or curved two-dimensional locus or points. In flight simulation, a surface is usually a polygon.

Pixel - The smallest unit of video or color of which the view screen is divided into.

INTRODUCTION

Early visual simulators were used in conjunction with a dynamic motion platform to enable the user to simulate real world action such as flying an aircraft, driving a tank or tasks which are difficult, dangerous or expensive to undertake in real situation. These systems were complete with instrumention comparable to the actual cockpit of an aircraft or vehicle being simulated. The simulation of the flight dynamics and interface with the cockpit motion were very realistic but the out-of-window visual effect was missing. Some success was achieved with model boards where a camera was controlled by the dynamics computer and projected onto the screens around the pilot. Still the need for a fully computer-generated image was apparent[2].

Current Computer Image Generator (CIG) devices are used in simulators for pilots, astronauts, navigators, and military trainees. These devices are nearly identical but differ in the database s from which they generate simulated images. Visual flight simulation is by far the major applications of CIG.

Typical CIG hardware is composed of a general-purpose computer (frame I), a geometric processor (frame II), and a video generator (frame III). The update rate of these

components is usually at 30 or greater HZ rate (30 static scenes per second) in order to yield the effect of continous or jitter-free motion. A visual database for a simulator is a model of some region of the world. A model portrays not only the topological and geometric structure of objects and surfaces, but also their color and texture. It is the database that drives the visual imagery hardware.

Polygons are used to describe modeled surfaces rather than higher order parameteric equation. Surface definition can be improved by increasing the number of polygons; however, as the number of polygons increase, the computational load increases. In order to reduce the cost, compromises are made to increase realism with the use of color and surface texturing.

Color greatly enhances the intelligibility and perceived quality of an image. Color perception is a powerful mechanism for discriminating among objects which would otherwise appear identical.

Natural scenes are rich in texture. The human visual system relies heavily on texture cues to perceive the structure in a scene. Realism of the desired surface is greatly enhanced by using texture pattern(s)[1]. For instance, it can simulate the visual sensations of air speed, height above the ground, tilt of the ground, location

in the environment, etc. Futhermore, texture provides additional cues for the relative size of objects and their relationship to one another[3].

Construction of texture patterns is done offline and need to be loaded into the CIG for review. The turnaround time has significant impact on debugging or improving new texture patterns. The objective of the IAT system is to efficiently accomplish that task.

CHAPTER I

TEXTURING

Basically, there are two classes of texture, standard/stripe and cell texture. Standard texture is a finite set of parallel stripes that get repeated until the entire area has been covered. Cell texture is the process of digitizing photos to give photographic realism[4]. This paper will not cover the discussion of cell texture since this is a feature of the Compuscene IV CIG. All texture discussions relate to Compuscene III.

Texture Code

Texture codes are used by the modeler for selecting a texture on certain polygons or faces. The basic parts of a texture code are listed below:

1. Pattern vector data
2. Modulation maps
3. Translation maps
4. Color code

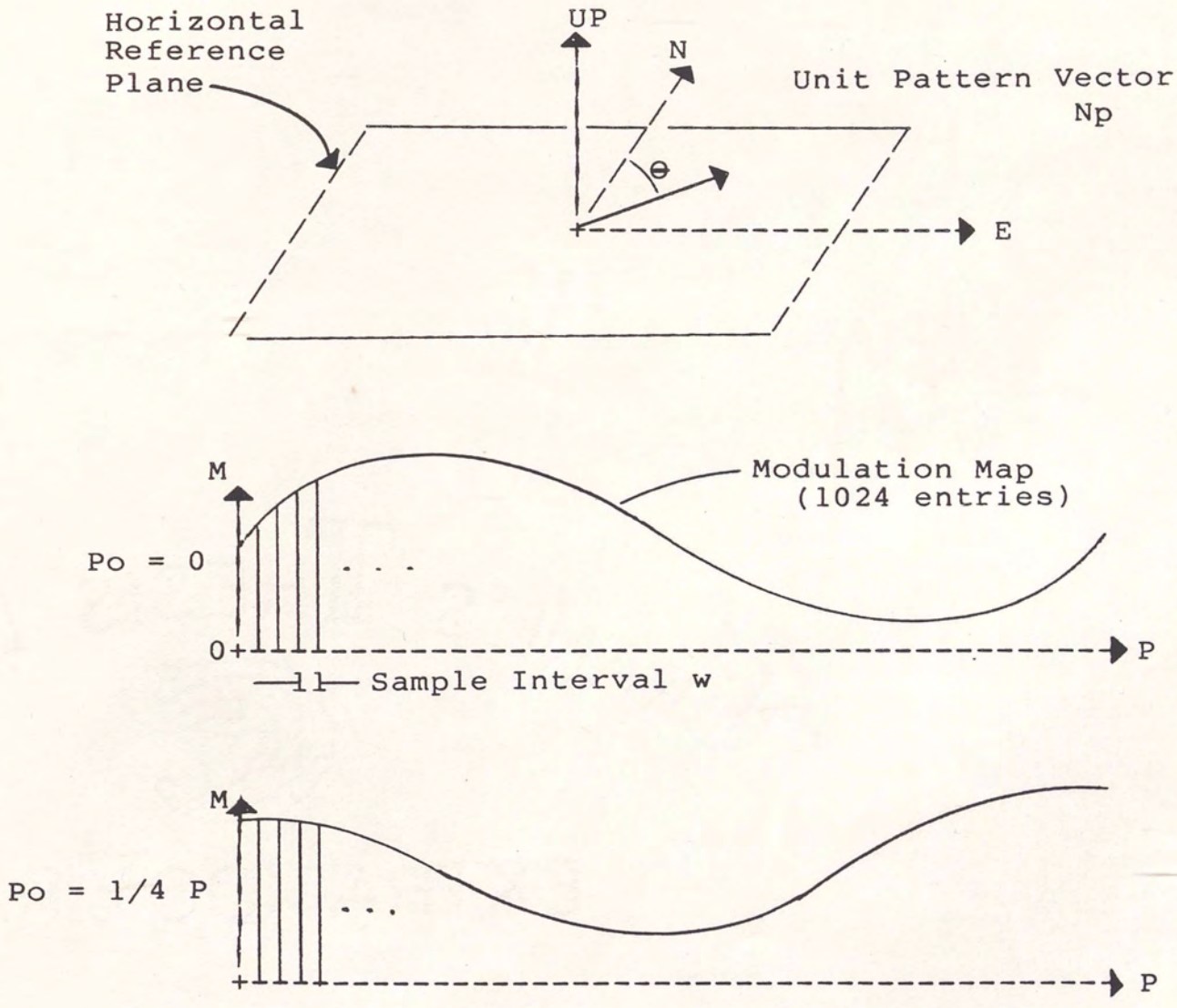


Figure 1. Pattern Vector Reference Data.

Pattern Vector Data

Pattern vectors define the direction and sampling interval/width of the selected texture modulation map. There are three required parameters for each modulation map used in defining a texture code. These are orientation angle, theta, scaling factor or width, and starting sample point P0, as shown in Figure 1, which will be discussed in greater detail in the requirements section.

Pattern vectors are defined in a reference plane and then projected onto the plane of individual faces in the hardware[5]. There are three reference planes as follows:

1. horizontal
2. vertical north-south
3. vertical east-west

The horizontal plane supports the texturing of 2D-features such as farmland, rivers, runway stripes, etc, whereas the vertical planes support the texturing of vertical or nearly vertical faces, such as the sides of a building, 3-D forest, and other 3-D features.

Pattern vector data is defined as follows:

pattern vector = KpNp, where

Kp = 1/1024w

w = user input sampling interval

theta = user input from keyboard

Np = unit pattern vector, a function of theta, environment or moving coordinate system, and reference plane. See Figure 2 for detail.

P0 = p/1024 0<= p(user input) <= 1024

Modulation Maps

Modulation maps are amplitude waveforms stored in digital form as one-dimensional tables in memory. These waveforms are shown in Figure 3. Each map consists of 1024 entries representing intensity values, varying from 0 (darkest) to 255 (brightest). A given map modulates the color intensity value of a specific face via the pattern vectors described earlier. The pattern vectors are projected into the plane of the calling face, which in turn causes the associated modulation map to modulate face pixel values in the direction and with a sampling factor specified by the pattern vector[6].

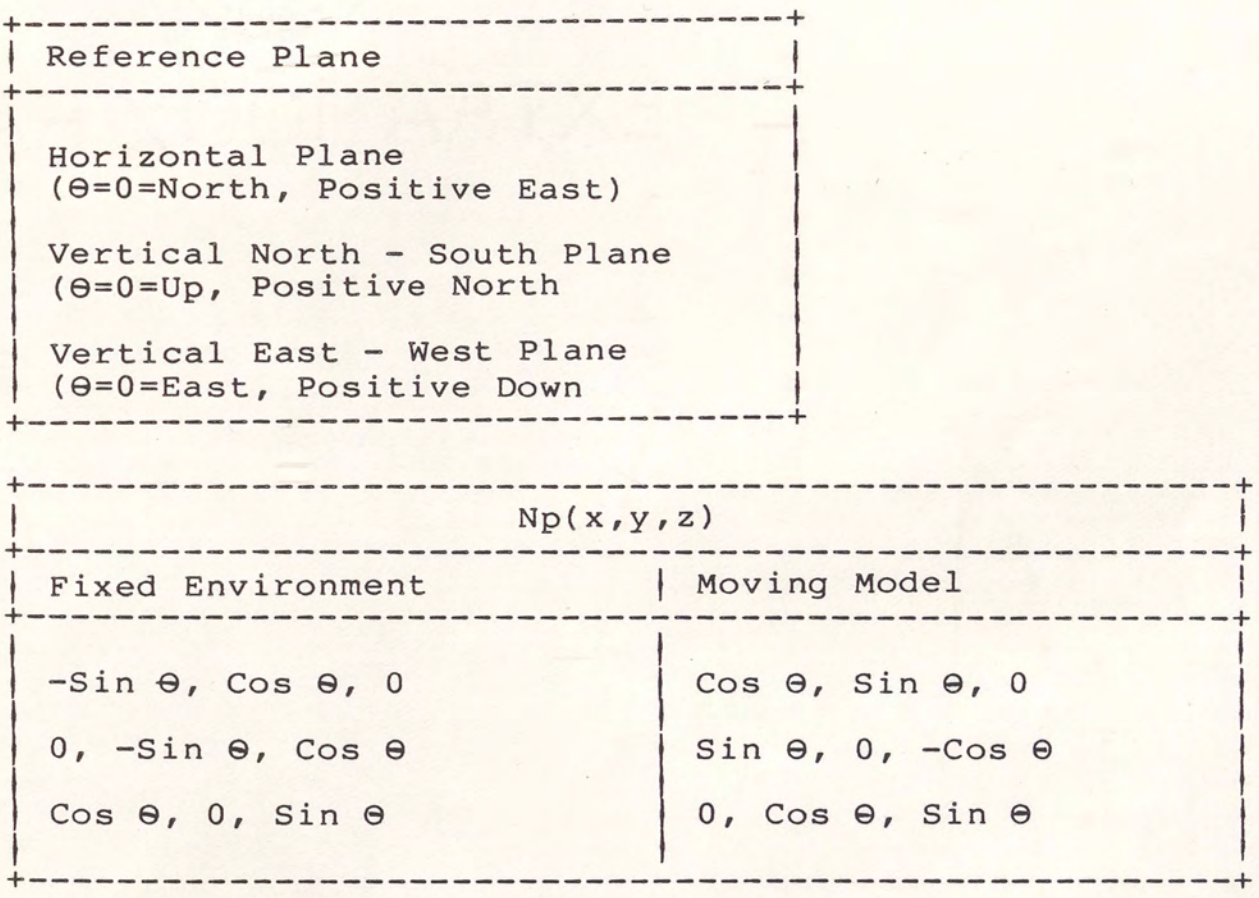


Figure 2. Definition of Environment, Moving Model Coordinate System, and reference Plane.

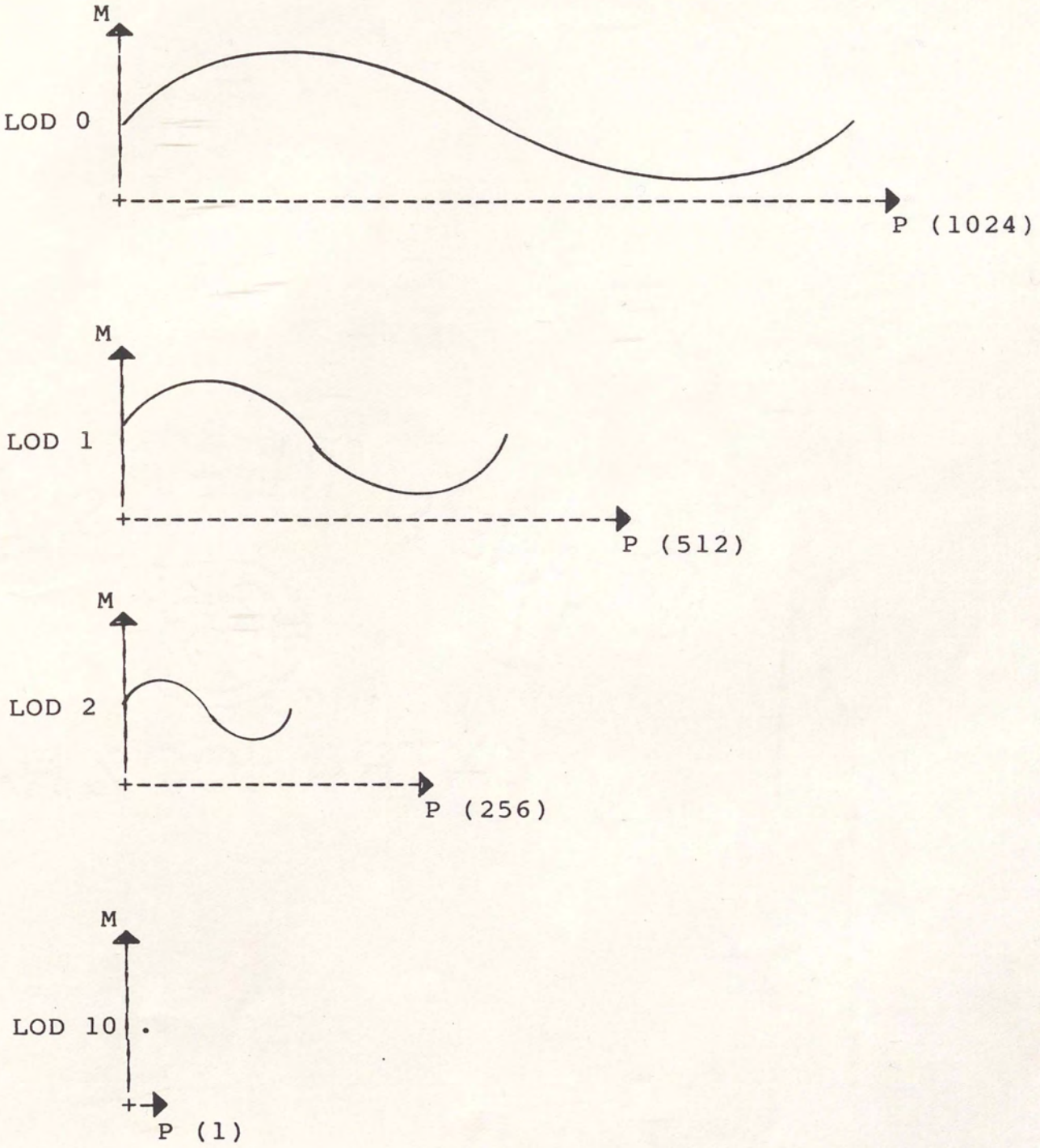


Figure 3. Example of Modulation Maps.

Modulation maps consist of a set of tables, one for each LOD implemented in the CIG. The number of entries at each LOD is one-half of the adjacent LOD, as shown in Figure 3 or 4, until the final LOD is reached whereby there is only one entry. The LOD decreases as distance increases to prevent an overload of texture edges in distance scenes. Similiarly, the amplitude range of the waveform decreases with LOD. The purpose is to reduce the noticeable contrast as range increases.

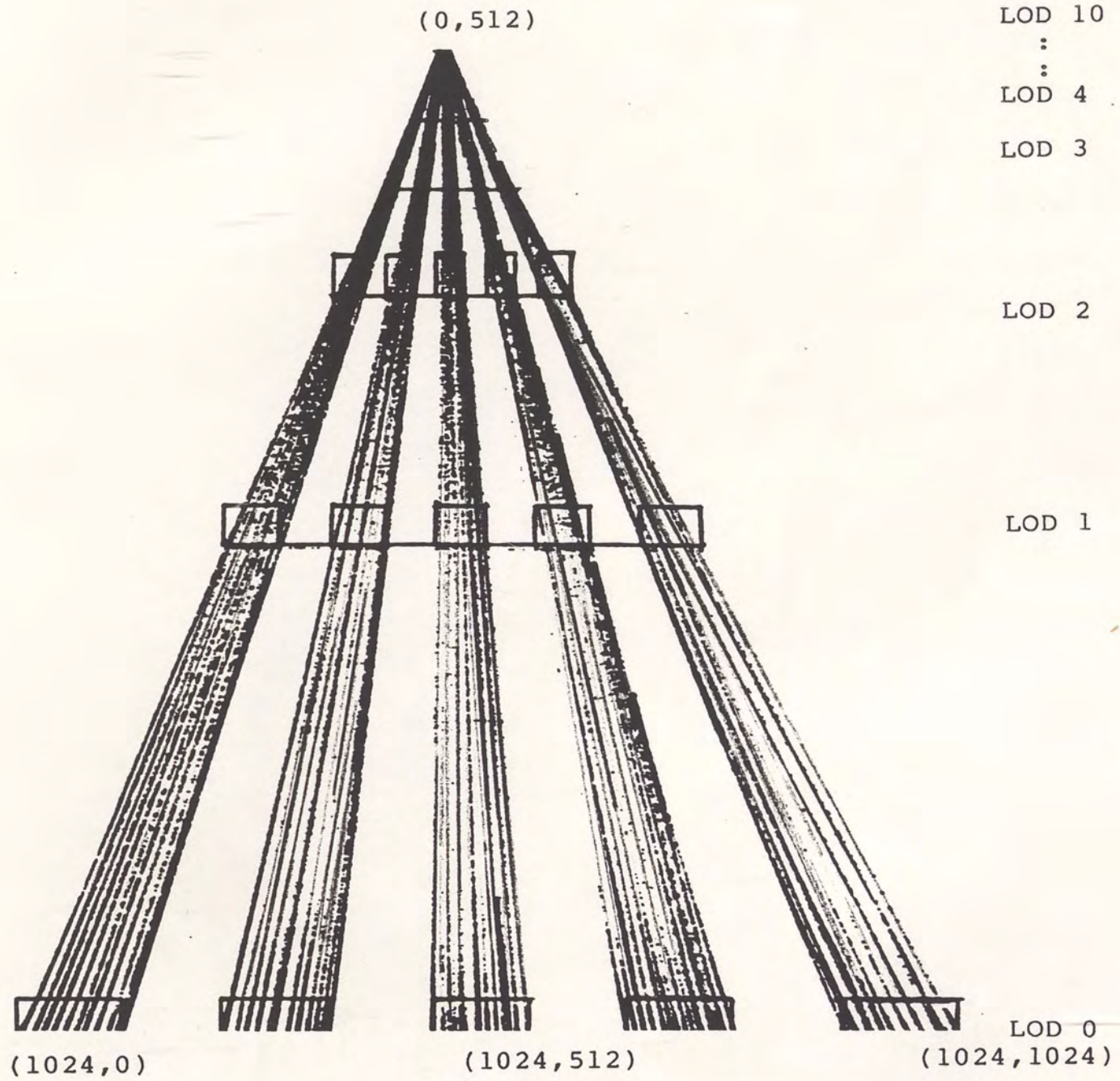


Figure 4. Diagram of LOD.

Translation Maps

Modulation maps are the basic building blocks of all texture patterns. But the CIG system can store only so many modulation maps, and the modulation maps must be modified to increase the variety of texture patterns. Translation maps are tables that used to increase the total number of texture patterns without redefining the modulation maps themselves. This is accomplished by taking a texture value computed by the texture generator and using it as a pointer to the translation table to look up the color intensity scale. A translation table can shift the brightness of the displayed colors up or down or filter out certain color effects. In other words, the translator map is an intensity transform function. The input and output ranges are from 0 to 255. The null translator map is a ramp with a slope of one. Figure 5 illustrates the translation tables for brightness and contrast adjustment.

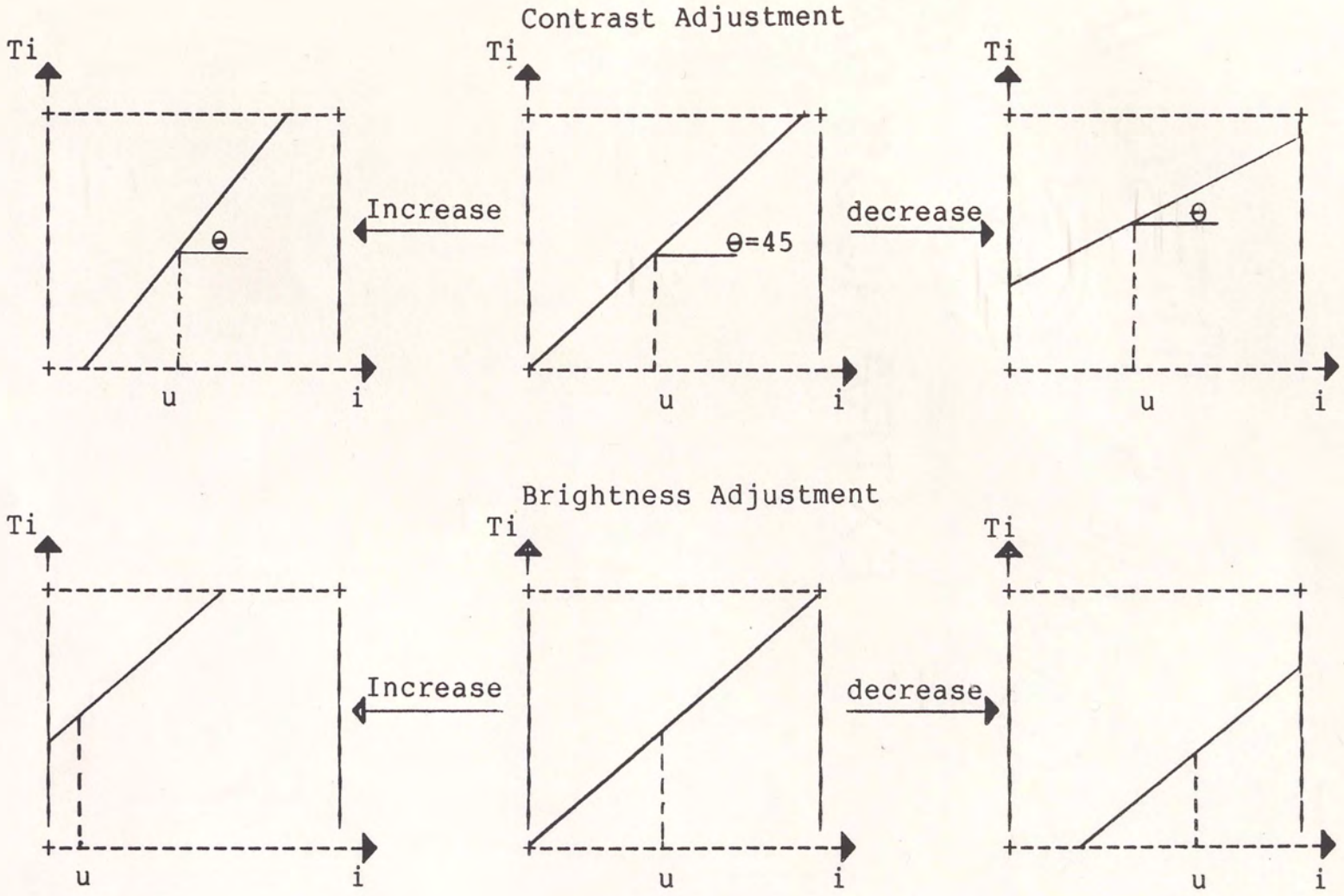


Figure 5. Translation Tables for Contrast and Brightness Adjustment.

Color Code

Color is used in conjunction with texture patterns to produce the desired effect. A great diversity of effects can be produced with a single texture pattern with the use of different colors. This is reflected in the use of texture codes and their use with different color codes which defines a location of stored red/green/blue color mixtures.

CHAPTER II

TEXTURE FUNCTION

Once texture pattern data has been defined and transferred to the CIG hardware, the texture generator will perform necessary functions to superimpose the selected texture pattern(s) upon the faces at the time the database is displayed on the visual devices. These functions are:

1. texture pattern generator function
2. texture pattern mapping
3. texture blending

The Texture Generator Functional Block Diagram is illustrated in Figure 6.

Texture Mapping Function

Texture patterns are mapped onto a polygon relative to the viewer. The orientation of the polygon determines the mapping characteristics, the property of compression and convergence is clear. The property of compression is found by finding the depth of the polygon. The property of convergence is found based on the orientation of the polygon normal[2]. When the size of the polygon decreases or the range of the polygon increases, the texture pattern will compress or squeeze together. When a polygon lies in a plane parallel to the viewer, a polygon will converge to a point.

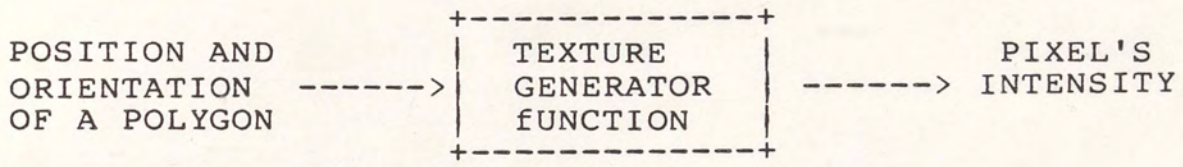


Figure 6. Texture Generator Functional Block Diagram.

There are two major functions in the texture generator. The first function is used to compute the texture value at any point on the polygon in terms of the view screen coordinate i and j[9], denoted

$$P = P0' + \frac{C1 + C2(I-I0) + C3(J-J0)}{C4 + C5(I-I0) + C6(J-J0)}$$

where:

$P0' = P0 - dot(KpNp,Vr)$

$C1 = KpNpu$

$C2 = KpNpwCw$

$C3 = KpNpvCv$

$C4 = Nfu/d$

$C5 = NfwCw/d$

$C6 = dot(Nf,Vr)$

$d = dot(Nf,Vr)$

P0', C1 - C6 are being calculated in the face modulation processor in Frame II. The input variables are texture starting value (P0), gradients (KpNp), viewpoint (Vr), face normal (Nf), and the constant relating vertical and horizontal field of view to the number of lines and elements, respectively[4].

The second function is to use this "P" as an address to look up the modulation function for texture intensity, i.e.,

$$M = g(P)$$

Hence, M is a functional of (i,j) in viewer space:

$$M = g[f(i,j)]$$

The texture modulation is then applied in the color times area principle to blend the different color representations within the pixel, to give the general representation of the pixel's intensity.

Texture Pattern Generator

The mapping function is used as an input to the texture pattern generator. The position and orientation of a polygon determines the appearance of the pattern. The pattern generator generates a variety of realistic patterns at different levels of detail without increasing the computational load.

There are different methods of realizing the texture pattern function. Dino Schwietzer suggested using textel as an individual texture element[7]. Blinn suggested using modulation of the reflection coefficient of the surface[8]. A more flexible technique for realizing the texture pattern generator is the table lookup technique. This method uses a table that contains texture intensity values. The output of the mapping function described below is used as an address to look up the texture intensity. This technique has some sound advantages. They are: flexibility in generating unlimited variations of the texture patterns requiring only one memory reference per sample point. The tradeoff of this single memory reference is that more high speed memory may be needed. It also requires the storage of multiple levels of detail tables and has impact on texturing blending implementation.

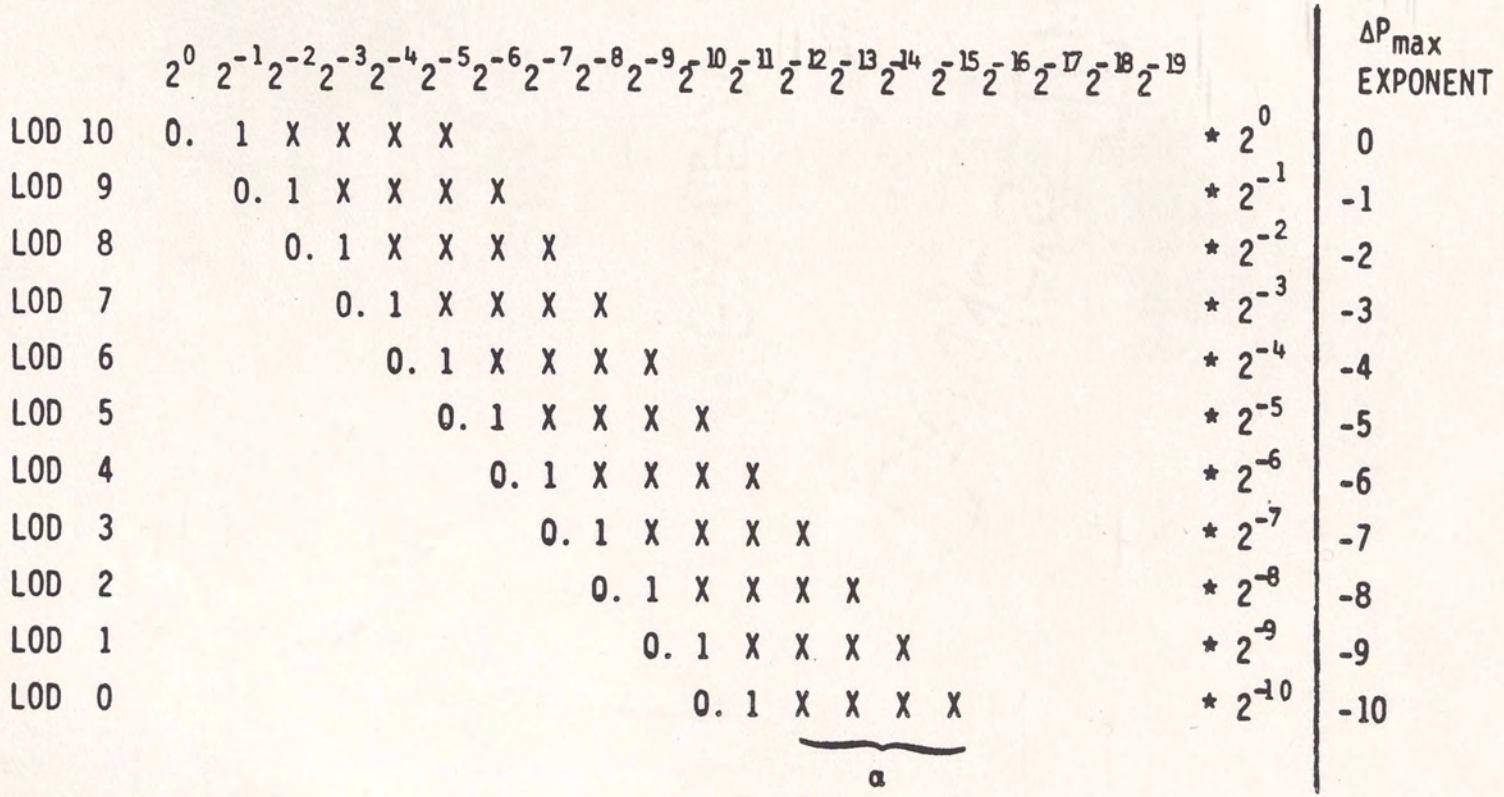


Figure 7. Definition of LOD

Texture Blending

Using a table lookup approach requires that texture patterns at different levels of detail can be stored to provide desired level of detail. Because of memory limitations, only a fixed and a finite number of levels of detail tables are stored. The effect of this is the transition from one level of detail to another becomes noticeable. Smoothing the transition between two levels of detail is done by blending.

Due to online memory limitations, Compuscene III uses ten LOD. The texture modulation value at the integer of level of detail (LOD0, LOD1, . . . LOD9) are stored. This LOD structure is shown in Figure 7. The modulation in between LODs is interpolated by the two adjacent LOD values, the current LOD and the next LOD values. Texture blending is a technique to obtain in between modulation value according to the deviation from the current level of detail.

The blending equation is:

$$m = aML + (1 - a)M$$

where: a is a 4-bit value, the interpolation is performed linearly and the range of a is the set denoted as

$$a = [1/16, 2/16, 3/16, 4/16, \ldots, 15/16]$$

m = modulation value

M = texture modulation at current LOD

ML = texture modulation at next LOD

Supposing that M = ML, then the blending equation reduces to M.

Referring again to Figure 6, LOD = 10 + exponent of delta-Pmax. The four significant bits (xxxx) are defined as a. It is clear that if a = 16/16, then the next LOD will be used. Also delta-Pmax will be catagorized as LOD.

CHAPTER III

INTERACTIVE TEXTURE

The implementation of the functions described in the requirements section, requires modification to the Real Time System (RTS) for Compuscene III. The IAT interface block diagram is shown in Figure 8. The Interactive Texture System is a separate task written to replace the Frame I Software package. This task has its own executive and command language as illustrated in Figure 9.

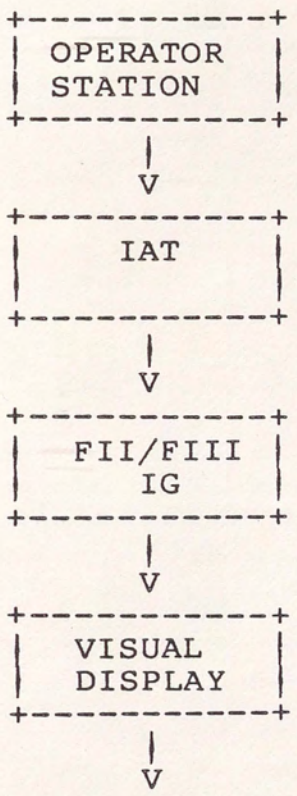


Figure 8. IAT Interface Block Diagram.

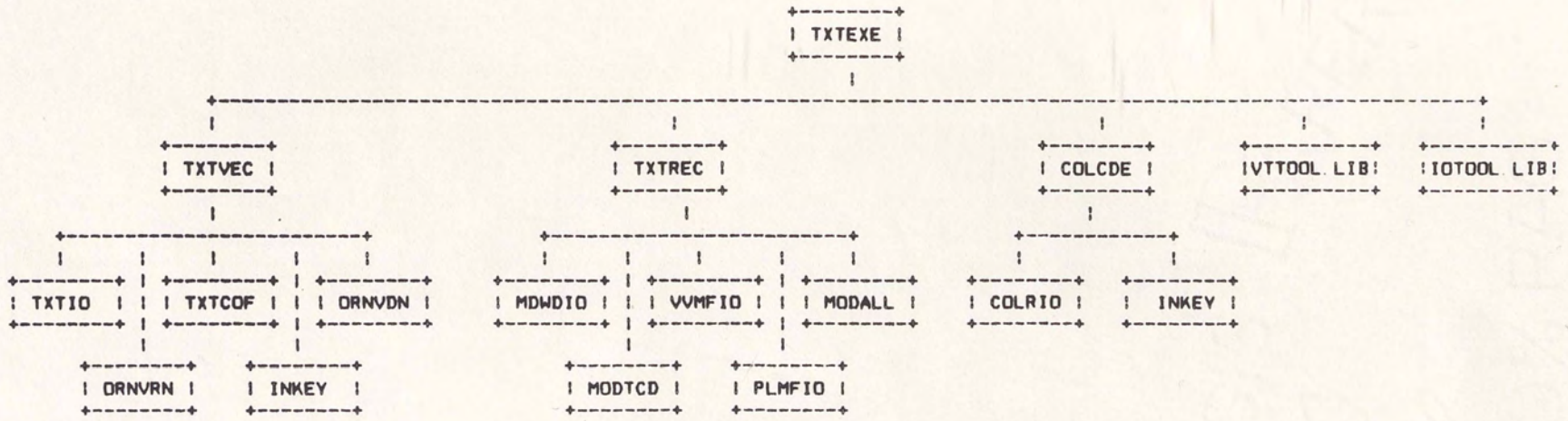


Figure 9. IAT Functional Block Diagram.

System Overview

The modeler uses the operator station and visual display to view texture patterns in a scene. The IAT system is used to examine/modify texture pattern data from this scene. Modified data is sent to the IG for viewing by the modeler. This process of varying texture pattern parameters is repeated until the modeler is satisfied with the scene. This procedure is shown on Figure 8.

Detailed Interface Requirements

The detailed interface block diagram is shown below:

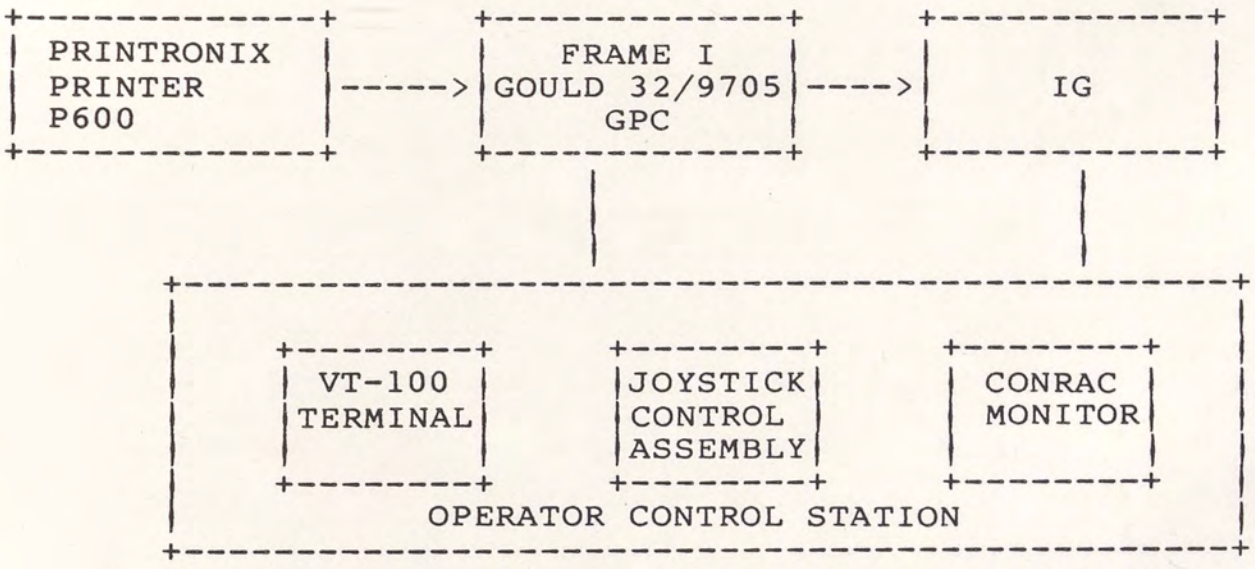


Figure 10. IAT Detailed Interface Block Diagram.

1. Hardware

The system configuration consists of a general purpose computer (Gould 32/9705), image generator subsystem, operator control station, and a Printronix printer P600. The general purpose computer interfaces to the IG and operator's station to provide the visual simulation. The joystick and CRT monitor interface with the IG for visual display and control. The printer is used to produce a hard copy for the user.

2. Support Software

- real-time software

The frame I real-time software consists of three tasks operating simultaneously; visual image processor (VIP), database update (DBU), and operator interface control (OIC). Their functions are briefly described as follows:

1. The VIP interfaces with the operator's station to generate images. All mathematical computations related to viewpoint and moving model positions, coordinate sets rotation, window defintion, visibility condition, etc. are performed by the VIP.

2. The DBU task provides all communication services between the disk and other frame I tasks. It processes requests to transfer terrain and model coordinate set EDB files to the CIG, and to transfer general data table files to the CIG.

3. The OIC task allows the operator to communicate to the IG through the shared memory of the frame I software. The OIC also displays selected variables from the shared memory.

- Database Generation System

The database generation system is composed of a set of offline tasks which provide the capability of creating and/or modifying Compuscene III visual database s.

There are three general types of data with which the Interactive Texture System is concerned.

a. Pattern Vector Data

This data is comprised of the parameters which are used together to compute the texture pattern's orientation and size.

b. Texture Record Data

This data is comprised of the parameters which for a given face, describe its Texture Code Address, RGB modulation, Face Color, Next Face Color, Blend Flag and Blend Select.

c. Color Code Data

This data is comprised of the parameters which define the Red/Green/Blue mixture of a given Color Code. There are 12 color code tables; 6 for Face Colors and 6 for Light Colors. Each table has 256 Color Codes and each Color Code is comprised of 1 Red/Green/Blue mixture.

IAT performs three major functions described as follows:

1. Examine/modify pattern vector file

2. Examine/modify texture records stored within database

3. Examine/modify color code mixtures files

Examine/Modify The Pattern Vector File

The Pattern Vector File contains Parameter information necessary to compute the Texture Pattern that will be displayed on a face.

The data stored in Pattern Vector File is outlined below:

a. Orientation Angle (or Orientation Components) - This defines the angle of the (stripes of the) Modulation Map. The angle is defined in three parts with three vector components, and they are measured from East which is defined as the X axis. The three vector components are labeled "X," "Y," and "Z." These three components are interdependent, and require modification as a group rather than independently.

b. Scaling Factor (or Spacing) - This defines the distance in feet between the Sample Points as stored in the Modulation Maps. A scaling factor of one

foot will produce a Texture Pattern that will extend 1024 feet before repeating (one foot times 1024 Sample Points = 1024 feet). The nature of this component requires that it be modified independently.

c. Starting Sample Point (or Phase Amplitude or Start Amplitude) - This component defines which Sample Point to start with in the Modulation Map. This starting position is registered at the block origin. The nature of this component requires that it be modified independently.

1. EXAMINE THE PATTERN VECTOR FILE

A single command entered with the keyboard will request the examination and/or modification of the components of a specified Pattern Vector address. The command will consist of a Pattern Vector Address, and a series of optional parameters. The optional parameters represent the Orientation Angle components, the Scaling Factor, and the Starting Sample Point. If the command is entered only with the address, the contents of this address will be displayed on the terminal screen. If the command is entered with the address and specified values for the parameters, the memory will be modified and the new values displayed on the terminal screen.

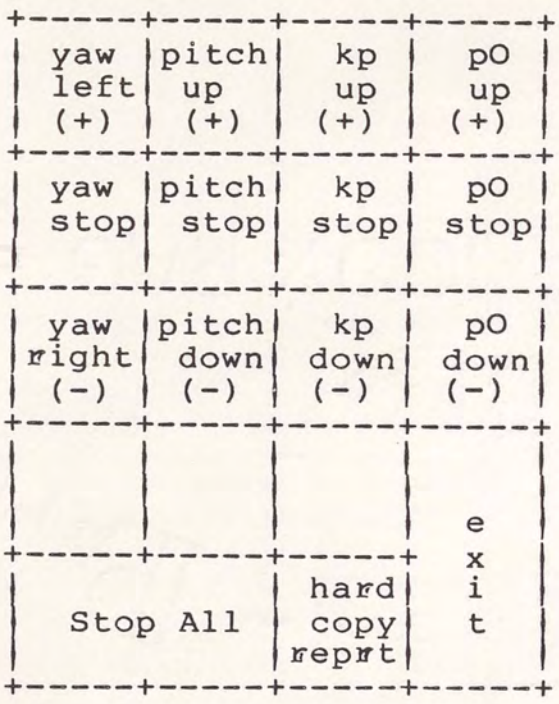
When the user requests the keypad option, the contents of the components will be displayed in fixed locations on the terminal screen.

2. MODIFY THE PATTERN VECTOR FILE

There will be two methods of modifying the components of a specified Pattern Vector Address. A single command will be available to enter known values. A second method using the technique of a programmable keypad will be available to dynamically change the values of the components. This will allow value experimentation for the components.

Modify Data With Single Command - A single command entered with the keyboard will request the examination and/or modification of the components of a specified Pattern Vector address. The command will consist of a Pattern Vector Address, and a series of optional parameters. The optional parameters represent the Orientation Angle components, the Scaling Factor, and the Starting Sample Point. If the command is entered only with the address, the contents of this address will be displayed on the terminal screen. If the command is entered with the address and specified values for the parameters, the memory will be modified and the new values displayed on the terminal screen.

Modify Data With Programmable Keypad - A single command entered with the keyboard will request that the programmable keypad be invoked. A single parameter necessary for execution will be the Pattern Vector Address. The contents of the components will be displayed in fixed locations on the terminal screen, and will be updated as the user modifies them with the keypad. The layout of the keypad is shown below:



Examine/Modify Texture Records Stored Within A Database

The Database contains a series of records for each face defined. One of these records contains the Shading, Texturing and Coloring information. (This record is called Vertex Vector Memory Face Data Storage). It is desirable to modify the information contained in this record in two

modes. The user can select a specific face and modify that Texture Record, or the user can select a specific Texture Code Address (T-Code) and modify all Texture Records matching the selected Texture Code. The Texture Record contains three types of fields. Each type of field will be modified separately. (This record contains many other flags of information. It is highly desirable that all the flags be modifiable, but for this application only the modification of Texture Pattern and Color information is required). The Texture Pattern and Color Data stored in these fields are outlined below:

a. Modulation Color Select - This is a three-bit field that indicates to the hardware which of the three color components to modulate. This information is initially defined with the Texture Code in the source file of the Database. This feature gives control to the modeler for manipulation of the red/green/blue color components without having to modify the Color Code Mixture file.

b. Texture Code Address - This is a twelve-bit field that is an address into a Texture Code Prom. This Prom contains records of addresses into the three special files needed for computation of the Texture Pattern. (The special files are: Modulation Maps File; Translation Tables File; and Pattern Vector File.) It is important that this address be modified to point to

another record in the Prom to select a different set of files. This function must be available on an individual basis, and also as global to the Database.

(This field is also the Shading Coefficient Address for faces that are flagged as Curved Surface Shaded. A separate flag will define what type of address is stored in this field).

c. Face Color Data - This information is contained in four fields, which represent two addresses and two flags. The two addresses are eight-bit fields for the face (Edge or Light) Color and the Next Face (Blending) Color. The two flags are a one-bit field for the Blend flag and a two-bit field for the Blend Select flag.

1. EXAMINE/MODIFY THE TEXTURE RECORD DATA

The technique of addressing the Texture Code Records stored within a Database is to find a specific face with a specific address, or to examine and/or list all faces that have a specific Texture Code Address. Once an address(s) has been identified, different commands will examine/modify the data stored within the record.

2. FIND/MODIFY A SPECIFIC FACE WITH A SPECIFIC ADDRESS

A specific face will be identified by entering a single command on the keyboard with a specific face address.

This address is called the Vertex Vector Memory Face Data address. This information will be obtained from an EDB plot and an EDB dump. The user will identify the desired face on the plot and find the address in the dump. Once a Face has been identified, two commands will be available to examine and/or modify the stored data. The first command will be used for Texture Pattern data, and the second command will be for Color.

Modify Texture Pattern Data - A single command will be entered with a series of optional parameters. These parameters represent the Modulation Color Select Fields (one each for the red/green/blue components), and the Texture Code Address. If the values of the parameters are left blank, the current contents will be displayed on the screen. If the command is entered with the face identifiers and specific values for one or more of the optional parameters, the memory will be modified and the new values displayed on the terminal screen.

Modify Color Code Data - A single command will be entered with a series of optional parameters. These parameters represent the Face Color Code, the Next Face Color Code, the Blend Flag, and the Blend Select Flag. If the values of the parameters are left blank, the current contents will be displayed on the screen. If the command is entered with the face identifiers and

specific values for one or more of the optional parameters, the memory will be modified and the new values displayed on the terminal screen.

3. FIND/MODIFY ALL FACES WITH A SPECIFIC TEXTURE CODE ADDRESS

A list of all faces that use a specific Texture Code Address will be identified and listed by entering a single command on the keyboard. The command will consist of a single identifier (which represents a Texture Code Address) and optional parameters. These parameters represent the Modulation Color Select and a new Texture Code Address. If the command is entered with specific values for one or more of the optional parameters, the memory will be modified.

Examine/Modify The Color Code Mixture Files

There are twelve Color Code Mixture Files for Compuscene III. Six are for Face colors and six are for Light colors. Each file contains records of Color Codes and red/green/blue (RGB) component mixtures. It is desirable to examine and/or modify the RGB mixtures for each Color Code in each table.

1. EXAMINE A FACE COLOR RED/GREEN/BLUE MIXTURE

A specific Face Color Code will be identified by entering a single command on the keyboard. The command

will consist of an optional Face Color File number, a Color Code, and optional RGB mixtures. If only the Face Color File number and Color Code are entered, the RGB mixture will be displayed on the terminal screen. If only a Color Code is entered, the RGB mixture for all six files will be displayed on the terminal screen. If the RGB mixtures are entered with a File number and a Color Code, the command will modify the memory and the new values displayed on the terminal screen. If the RGB mixtures are entered with only a Color Code, all six Face Color Files will be modified and the new values will be displayed on the terminal screen.

When the user requests the keypad option, the RGB mixture will be displayed in fixed locations on the terminal screen.

2. EXAMINE A LIGHT FACE COLOR RED/GREEN/BLUE MIXTURE

A specific Light Face Color Code will be identified by entering a single command on the keyboard. The command will consist of an optional Light Face Color File number, a Color Code, and optional RGB mixtures. If only the Light Face Color File number and Color Code are entered, the RGB mixture will be displayed on the terminal screen. If the RGB mixtures are entered with a File number and a Color Code, the command will modify the memory and the new values displayed on the terminal

screen. If the RGB mixtures are entered with only a Color Code, all six Light Face Color Files will be modified and the new values will be displayed on the terminal screen. If the RGB mixtures are entered with only a Color Code, all six Light Face Color Files will be modified and the new values will be displayed on the terminal screen. When the user requests the keypad option, the RGB mixture will be displayed in fixed locations on the terminal screen.

3. MODIFY FACE COLOR MIXTURE WITH SINGLE COMMAND

There will be two methods of modifying the RGB mixtures of a specified Color Code. A single command will be available to enter known values. A second method using the technique of a programmable keypad will be available to dynamically change the values of the components. This will allow experimentation to determine the desired values of the components.

A Specific Face Color Code will be identified by entering a single command on the keyboard. The command will consist of an optional Face Color File number, a Color Code, and optional RGB mixtures. If only the Color File number and Color Code are entered, the RGB mixture will be displayed on the terminal screen. If only a Color Code is entered, the RBG mixture for all six files will be displayed on the terminal screen. If

the RGB mixtures are entered with a File number and a Color Code, the command will modify the memory and the new values displayed on the terminal screen. If the RGB mixtures are entered with only a Color Code, all six Face Color Files will be modified and the new values will be displayed on the terminal screen.

4. MODIFY LIGHT FACE COLOR MIXTURE WITH SINGLE COMMAND

There will be two methods of modifying the RGB mixtures of a specified Color Code. A single command will be available to enter known values. A second method using the technique of a programmable keypad will be available to dynamically change the values of the components. This will allow experimentation to determine the desired values of the components.

A specific Light Face Color Code will be identified by entering a single command on the keyboard. The command will consist of an optional Light Face Color File number, a Color Code, and optional RBG mixtures. If only the Color File number and Color Code are entered, the RBG mixture will be displayed on the terminal screen. If only a Color Code is entered, the RGB mixture for all six files will be displayed on the terminal screen. If the RGB mixtures are entered with a File number and a Color Code, the command will modify

the memory and the new values displayed on the terminal screen. If the RGB mixtures are entered with only a Color Code, all six Face Color Files will be modified and the new values will be displayed on the terminal screen.

5. MODIFY A COLOR MIXTURE WITH PROGRAMMABLE KEYPAD

A single command entered with the keyboard will request that the programmable keypad be invoked. The parameters necessary will be the Face (or Light Face) Color File and Color Coded. The RGB mixture will be displayed in fixed locations on the terminal screen, and will be updated as the user modifies them with the keypad. This command will be able to modify any of the six Face Color Files or six Light Face Color Files. The layout of the keypad is shown in Figure 11 below:

This keypad layout will allow the user to use two different systems for modifying a color mixture. The first is based on the actual Red/Green/Blue color mixture values. The user will manipulate the values separately. The second system allows the modification of the Red/Green/Blue color mixture values based on the Hue, Saturation, and Intensity. These systems will be available both together and separately.

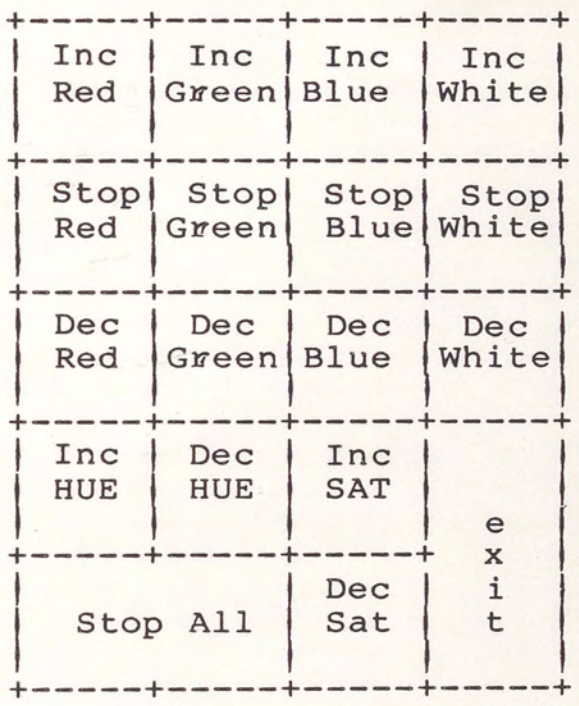


Figure 11. Color Mixture Keypad Layout.

1. Interactive Texture Commands

The following rules apply for all commands.

1. COMMAND EXECUTION

All commands are entered at the Executive level of execution thus making data manipulation very rapid. The exception is when the keypad is in a Dynamic Update mode, the exit key must be typed. This will be discussed later.

2. COMMAND FORM All commands have the form:

COMMAND-DESCRIPTOR PARAMETER1, . . . PARAMETERn

The COMMAND-DESCRIPTOR indicates which type of Texture Data is being examined or modified and the mode of operation. There are two modes of operation, Interactive and Dynamic.

In the Interactive mode the command is executed as a one-time request and the results appear immediately on the CRT and Viewpoint Monitor. In the Dynamic mode the keypad becomes active and accepts dynamic input. The resulting effects in this mode are continually updated to the CRT and Viewpoint Monitor in Real-Time.

3. COMMAND PARAMETERS

There are two types of parameters, IDENTIFIERS and

UPDATERS. Identifiers are used to locate the current data to be modified and updaters are the actual values used to replace the current data.

Selected parameters may be skipped:

COMMAND-DESCRIPTOR PARAMETER1, . . . PARAMETER4

In this example parameters 2 and 3 were skipped and will remain unchanged whereas parameters 1 and 4 will be modified. In the Dynamic Mode all UPDATE parameters must be skipped.

4. SPECIFIC DATA TYPE INSTRUCTIONS

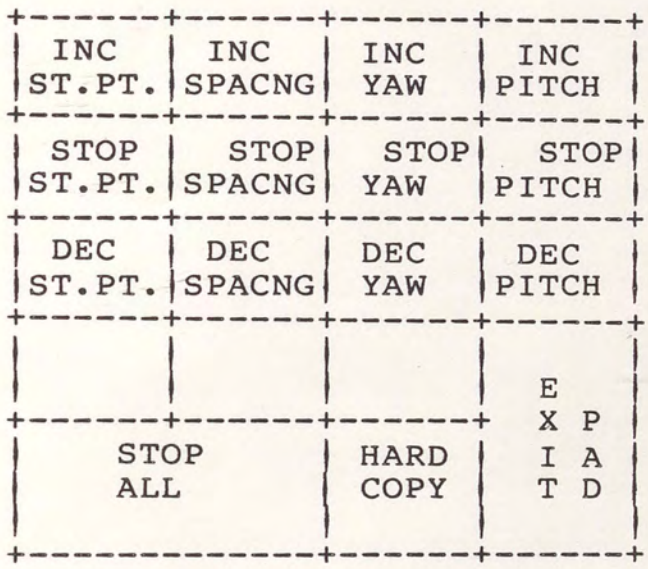
The following sections define the Command Descriptors and legal parameters for the three different data types.

PATTERN VECTOR EXAMINE-MODIFY ASSIGNMENT

OPTIONS:	EFFECT
PVA pn	examine pattern vector data
PVA pn,st,sp,ya,pi	modify pattern vector data
PVD pn	dynamically modify pattern vector data (keypad mode)

WHERE:
pn = Pattern Number 1-6. Not skippable (NS).
st = Start Sample Point (.125-1024). Skippable (S).
sp = Spacing (0-1024). S
ya = Yaw (0-360). S
pi = Pitch (-90 to 90). S

KEYPAD LAYOUT



TEXTURE RECORD EXAMINE-MODIFY ASSIGNMENT

OPTIONS:	EFFECT
TRA cs,va	examine one texture records data
TRA cs,va,tc,rd,gr,bl	modify one texture records data
TRG cs,cc,tc,rd,gr,bl	modify all texture record data for matching texture-code-adr
TCA cs,va	examine one texture color's data
TCA cs,va,fc,nc,bf,bs	modify one texture color's data

WHERE:

cs = coordinate set number (1-24). NS
va = vertex vector memory adr(vvm adr from edb dump) NS
rd = modulation color select red component (0 or 1). S
gr = modulation color select green component (0 or 1). S
bl = modulation color select blue component (0 or 1). S
tc = texture code address (0-2047). S
fc = face color code (0-255). S
nc = next face color code (0-255). S
bf = blend flag (0 to 1). S
bs = blend select (0-3). S
cc = current texture code adr(to be searched for and modified by parameters). NS

```
+---------------+
|    KEYPAD     |
|   NOT USED    |
+---------------+
```

COLOR CODE EXAMINE-MODIFY ASSIGNMENT

OPTIONS:	EFFECT
FCA ft,cc	examine face color data
FCA ft,cc,rd,gr,bl	modify face color data
LCA lt,cc	examine light color data
LCA lt,cc,rd,gr,bl	modify light color data
FCD ft,cc	dynamically modify face color data (keypad mode)
LCD lt,cc	dynamically modify face color data (keypad mode)

WHERE:

ft = face table number (1-6). NS
lt = light table number (1-6). NS
cc = color code table entry (1-256). NS
rd = color code red component (0-255). S
gr = color code green component (0-255). S
bl = color code blue component (0-255). S

KEYPAD LAYOUT

```
+-----+-----+-----+-----+
| INC | INC | INC | INC |
| RED |GREEN|BLUE |WHITE|
+-----+-----+-----+-----+
| STOP| STOP| STOP| STOP|
| RED |GREEN| BLUE|WHITE|
+-----+-----+-----+-----+
| DEC | DEC | DEC | DEC |
| RED |GREEN|BLUE |WHITE|
+-----+-----+-----+-----+
| INC | DEC | INC | E   |
| HUE | HUE | SAT | X P |
+-----+-----+-----+ I A |
|    STOP     | DEC | T D |
|     ALL     | SAT |     |
+-------------+-----+-----+
```

QUALITY EVALUATION

The last step in the software development cycle is to determine whether a system has satisfied the program's goal and quality. This paper will be using reliability, maintainability, availability, portability, performance, and human engineering as quality requirements.

1. RELIABILITY

The program produces consistent output for the same input. All computations use floating point to avoid truncation as well as to enhance accuracy. The program's accuracy is valid up to plus or minus ten raised to thirtieth power.

2. MAINTAINABILITY

Since the program is modularized and structured, the flow of control is easy to follow. The program is very self-descriptive, and communicative. In other words, the program's clarity, readability, and understandability is clear because each module has its objective, constraints, input, output, and calling relationship spelled out. As a result, modification and maintainability is made easy.

3. AVAILABILITY

The goal of this program is to provide flexibility to

all users. Hence, the program is designed to reside permanently as the real-time software for easy access.

4. PORTABILITY

Since real-time software is written in Fortran, the program is implemented using Fortran to avoid possible system type of errors. Machine-dependence will be reduced if good programming techniques are followed in manipulating the word size and character set.

5. PERFORMANCE

The keypad option reduces the amount of interaction required by the modeler for modification to texture pattern data by providing a convenient method of control and data entry. The program also reduces the turnaround time to debug or improve texture pattern data of an actual scene without repeatedly modifying, loading, and testing the database s.

6. HUMAN ENGINEERING

The program is user-friendly as the program accepts and echoes invalid data. In addition, most parameters are skippable or optional, thus reducing modeler's data entry input and subsequent input errors.

LIMITATIONS

It is important for a programmer or user to understand the limitations of a program. Understanding what a program cannot do is as important as knowing what it can do.

1. Keypad option only operates on VT100-VT200 series keyboard.

2. After changes have been made to texture, user must go back to texture load files, EDB files, vector files, etc., and make modifications. The data must then be returned through DBGS software.

3. After changes have been made to color, user must go back to run COLSYS program to change the red, green, and blue values assigned to the current number in the color files.

4. IAT is unable to trace a face or point feature characteristics from the display back to EDB from which it originated.

5. After operations on the selected scene, the database must be reloaded so that the real time system can continue.

CHAPTER III

CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

This paper has shown that the IAT system is an effective tool for the database modelers to verify or improve texture patterns. The IAT system eliminates the costly time consuming procedures of modifying, loading, and testing the database for desired texture patterns. In addition, this paper describes how texturing enhances realistic visual scenes relative to the viewer and the implementation technique. Finally, the discussion of the texture data such as pattern vector data, modulation maps, translation maps, and code data is intended to give better insight into the IAT command operations on these elements.

The IAT system can be made more efficient by eliminating the limitations mentioned earlier. Some suggestions for future enhancement are listed below:

1. Allow the user to resume real time system processing after completing the IAT function.

2. Identify a face or point feature using the joystick and generate a cursor symbol '+' on the feature.

3. Identify EDB and location within EDB from which the point or feature originated.

APPENDIX

Program Detail Design Documentation

Title: TXTEXE - Interactive Texture Executive

Functional Description:

Selects via operators commands to perform online modification Pattern Vector data, Texture Records, or Color Codes in the Frame II memory.

The program design language for CPC TXTEXE is given below:

```
Program TXTEXE
Initialize screen windows, and Frame I common
Prompt user for command for I/A Texture Function
Do while not an exit command
   Input command from keyboard
   If command equals PVA, then
        Disable Keypad
        Call TXTVEC Texture Pattern Vectors modification
   Elseif command equals PVD, then
        Enable Keypad
        Call TXTVEC Texture Pattern Vectors modification
   Elseif command equals TRA, then
        Disable color
        Disable allrec
        Call TXTREC Texture record(s) modification
   Elseif command equals TCA, then
        Enable color
        Disable allrec
        Call TXTREC Texture record(s) modification
   Elseif command equals TRG, then
        Disable color
        Enable allrec
        Call TXTREC Texture record(s) modification
   Elseif command equals FCA, then
        Disable light
        Disable Keypad
        Call COLCDE Color Code examine/modification
```

```
   Elseif command equals LCA, then
        Enable light
        Disable Keypad
        Call COLCDE Color Code examine/modification
   Elseif command equals FCD, then
        Disable light
        Enable Keypad
        Call COLCDE Color Code examine/modification
   Elseif command equals LCD, then
        Enable light
        Enable Keypad
        Call COLCDE Color Code examine/modification
   Else
        Display "INVALID COMMAND"
   Endif
   Prompt for next option command
Enddo
End
```

Title: TXTVEC - TEXTURE PATTERN VECTORS MODIFY MODULE

Functional Description:

Perform online modification of texture pattern vector parameters for the selected pattern number.

The program design language for CPC TXTVEC is given below:

```
Subroutine TXTVEC

Erase the data display and error regions on the VT-100
Extract pattern vector number from input
Determine if pattern vector number is valid
Create index to 'TXCOEF'
Get normal pointer from 'TXCOEF' to point to normals in 'TXNORM'
Initialize Delta variables
Initialize first pass flag
If dynamic mode (Keypad option) then
   Initialize the examine/modify flag
   Display header on VT-100
   Get KP and Po from 'TXCOEF'
   Get NPX,NPY,NPZ from 'TXNORM'
   If a keypad key was depressed, do requested function
Endif
Denormalize NPX,NPY,NPZ Add delta values then Renormalize
Denormalize KP Add delta values then Renormalize
Denormalize Po Add delta values then Renormalize
Store KP,Po in 'TXCOEF'
Store NPX,NPY,NPZ in 'TXNORM'
Call Real time Dynamic Update routine
If this parameter is not a skip and if legal value, then
   Renormalize param to find KP and store in RTS table
Endif
Save NPX,NPT,NPZ current values from RTS table
Get Yaw and Pitch
Get Yaw from inbuf
IF not skip and if legal value, then
   Denormalize curreny NPX,NPY,NPZ to determine Yaw
   Denormalize to determine Yaw
   Validate Yaw entered
Endif
Get Pitch from inbuf
IF not skip and if legal value, then
   Denormalize curreny NPX,NPY,NPZ to determine Pitch
   Denormalize to determine Yaw
   Validate Pitch entered
Endif
Store Working pitch in pitch
Store working yaw in yaw
```

```
Renormalize Yaw and pitch and store resulting NPX,NPY,NPZ
   in RTS table 'TXNORM'
If this is an Update request
   Set up input parameters for 'TXCOEF'
   Call TXTCOF to update coefficient data
   Call TXTIO to write data to Texture Generator
Endif
Get KP, Po and denormalize to find W,Ps
Get NPX,NPY,NPZ and denormalize to find Yaw & Pitch
Report requested data
Clear scrolling region and reset cursor
Return
End
```

Title: TXTREC - Texture Record(s) Modify Module

Functional Description:

Perform online modifications of a texture record for a selected face or for allfaces within a EDB.

The program design language for CPC TXTREC is given below:

```
Subroutine TXTREC

Inhibit Frame II processing
Erase data display and error regions on VT-100
Start extracting data from the input buffer inbuf
Read the selected texture record data from the VVM face data
If Color data examine/modify request, then
   Get and validate color code from inbuf
   Get blend data from inbuf
Endif
If texture code request, then
   Get texture address from inbuf
   Get Red modulation color select from inbuf
   Get Green modulation color select from inbuf
   Get Blue modulation color select from inbuf
   If this is a modify request, then
      Update VVMBUF and write update array to I.G.
   Endif
Endif
If modify all texture records request, then
   Call Modall to modify all texture records in VVM for
   specified CS#
Else
   Report data to VT-100 screen
Endif
Clear scrolling region and position cursor
Return
```

Title: COLCDE - Color Code Examine/Modify Module

Functional Description:

Perform online modification of the light and face table color codes in the Frame II color memory.

The program design language for CPC COLCDE is given below:

```
Subroutine COLCDE
Call INPSCN to get Table number from input buffer
Call INPSCN to get Color Code from input buffer
Call INPSCN to get Red Color from input buffer
Call INPSCN to get Green Color from input buffer
Call INPSCN to get Blue Color from input buffer
If light is true, then
   Initialize the color update light table with
     sync data, control words, and color data
Else ( Face data )
   Initialize the color update face table with
     sync data, control words, and color data
Endif
If Keypad is enabled, then
   Print Face/Light Color Code Red Green Blue header on screen
   Do while keypad code is not escape
      Sample keypad and update delta values accordingly
      If first pass thru keypad, then
         Extract color data (red, grn,blu)
      Endif
      If Delta Values for Hue, intensity, or Saturation are
         non-zero, or first pass thru keypad, then
         Denormalize red, grn, and blu to produce hue,intensity
         and saturation value for delta values
      Endif
      If delta value for red, grn, and blu are non-zero, then
         Modify red, grn, and blu using delta values.
      Endif
      If any delta values are non-zero,then
         Update dynamic memory
      Endif
   Enddo
   Reset first pass flag
Else examine is enabled, then
   Display color values on screen
Else update color (red, grn, or blu) data from parameters
   Update dynamic memory
Endif
Display new color data on screen
Return
```

Title: COLRIO - Transmit Color data block to IG

Functional Description:

This routine sets up the necessary control for color data transfer. It also updates the Face or Light data upon request.

The program design language for CPC COLRIO is given below:

```
Subroutine COLRIO

Initialize memory addresses
Set up control words
If face color, then
   Copy memory address to transmit buffer
   Copy Face color data to transmit buffer
   Copy light color data to transmit buffer
Endif
Compute working buffer index
Update red, grn, blu color
Inhibit Frame II processing
Call MDWDIO to save mode word
Call MDWDIO to restore mode word
Set up control word to read sync
Set up conrrol word for data transfer
Write data to dynamic memory buffer
Report I/O errors
Return
End
```

Title: TXTCOF - Texture Coefficient Processor

Functional Description:

This routine creates texture coefficients from the patterns and thier normals

The program design language for CPC TXTCOF is given below:

```
Subroutine TXTCOF

If texture is enabled, then
   Clear flag and get viewpoint assigned
   Load viewpoint assigned
   If normal assigned to viewpoint, then
      Get index into TXCOEF nad TXNORM
      If not rotated, then
         Rotate it and drive dot product
      Endif
      Dot norm with position
      Apply scale for KP in Po calculation
   Endif
   Scale and store Po,C1,C2,C3
Endif
Return
End
```

Title: TXTIO - Texture Coefficient data I/O module

Functional Description:

Inhibit Texture processing and move texture coefficients data to Texture Generator.Restore processing upon completion.

The program design language for CPC TXTIO is given below:

```
Subroutine TXTIO

Inhibit Texture processing
Set up IOCB for read sync
Transfer data to Texture Generator
Echo transfer data
Restore texture processing
Return
End
```

Title: VVMFIO - Vector Vertex Memory I/O Routine

Functional Description:

Performs Read/write of virtual blocks of data from/to the VVM memory in Frame II.

The program design language for CPC VVMFIO is given below:

Subroutine VVMFIO

```
Set up control word to read sync
Determine HSD read or write
If a write request, then
   Move data to transmit buffer
Endif
Set up control word for data transfer
Write data to Frame II memory
If a read request, then
   Put data into the return buffer
Endif
return
```

Title: MDWDIO - Mode word I/O module

Functional Description:

Perform read write of mode word data from/to the I.G.

The program design language for CPC MDWDIO is given below:

```
Subroutine MDWDIO

If read ,then
   Set read flag
Endif
Set up control words for read/write
Send control word to I.G.
If read, then
   Read control word from I.G.
Endif
Return
End
```

Title: MODTCD - Modify Texture Code Data

Functional Description:

Perform Modification of Texture Code records residing in the VVM face data array 'VVMBUF' using modification parameters to prepare the 'VVMBUF for transmission to the IG.

The program design language for CPC MODTCD is given below:

```
Subroutine MODTCD

If texture code address parammeter to be modified, then
   Put new value into output buffer VVMBUF
Endif
If Red Modulation color select to be modified, then
   Put new value into VVMBUF
Endif
If Green Modulation color select to be modified, then
   Put new value into VVMBUF
Endif
If Blue Modulation color select to be modified, then
   Put new value into VVMBUF
Endif
Return
End
```

Title: MODALL - Modify all texture records

Functional Description:

Given a CS and a texture code address, the VVM is sequentially searched and for each matching texture code address, a new specified texture code address and parameter are used to replace the old for a given CS .

The program design language for CPC MODALL is given below:

```
Subroutine MODALL

Set model block pointer
Output heading
Read the pointer list model header data (2 words)
Create index to vertex memory
Do for each face
   Read this face data from vertex memory
   If this face's texture code matches the compare
       texture code, then
      Call MODTCD to modify this faces texture code
      Write the modifued data back into vertex memory
   Endif
   Update face pointer
   Update (decroment face count) face count
Enddo
Return
End
```

Title: PLMFIO - Pointer List Memory I/O Routine

Functional Description:

Performs Read/write of virtual blocks of data from/to the PLM memory in Frame II.

The program design language for CPC PLMFIO is given below:

```
Subroutine PLMFIO

Set up control word to read sync
If a write request, then
   Move data to transmit buffer
Endif
Set up control word for data transfer
Write data to Frame II memory
If a read request, then
   Put data into the return buffer
Endif
return
```

Title: ORNVRN - Orientation vectors renormalize

Functional Description:

Renormalize Texture Pattern vectors.

The program design language for CPC ORNVRN is given below:

Subroutine ORNVRN

```
If yaw = 90 deg, yaw = yaw - .001
If yaw = 270 deg, yaw = yaw + .001
If pitch = 90 deg, pitch = pitch - .001
If pitch = 0 deg, pitch = pitch + .001
If pitch = -90 deg, pitch = pitch - .001
Convert pitch and yaw to radians
Renormalize to determine NZ where
NZ = SQRT(1/1/(tan(pitch))**2+1)
Renormalize to determine NX where
NX = SQRT(1-NZ**2)/(tan(yaw)**2)+1
Renormalize to determine NY where
NY = NX(tan(yaw))
If vector is in quadrant 2 or 3, then
   Take inverse values for NX,NY
Endif
If vector is negative ,then
   Take inverse value for NZ
Endif
Return
```

Title: ORNVDN - Orientation vectors denormalize

Functional Description:

Denormalize the Orientation vectors

The program design language for CPC ORNVDN is given below:

Subroutine ORNVDN

```
Denormalize NPX,NPY to determine yaw
Yaw = ARCTAN (NPY / NPX)
Convert yaw from radians to degrees
If vector in quadrant 2 yaw = 180.0 - yaw
If vector in quadrant 3 yaw = 180.0 + yaw
If vector in quadrant 4 yaw = 360.0 - yaw
Denormalize NPX,NPY,NPZ to determine pitch
Pitch = ARCTAN (NPZ / SQRT(NPX**2+NPY**2))
Convert pitch from radians to degrees
Return
```

REFERENCES

[1] Foley, J. D., and Dam, A. V. 1982. Fundamentals of Interactive Computer Graphics, Reading, MA: Addison-Wesley, 1982.

[2] Baker, W.,Compuscene III Frame I Design,(G.E. Technical Report PIR C-III-663-WB-785) January, 1984.

[3] Schachter, B. J. 1983. Computer Image Generation. New York: Wiley and Sons, 1983.

[4] Miller, T. An Introduction of General Electric CIG's Use of Texture, (G.E. Technical Report PIR C-130-637-TEM-902) August, 1983.

[5] Ferguson, R. Texturing Modelling Guidelines, (G.E. Technical Report PIR 2363-606-REL-949) June 1983.

[6] Ferguson, R. Texture Table Generator, (G.E. Technical Report PIR 2363-606-REL-958) July, 1983.

[7] Schweitzer, D. 1983. Artificial texturing: An aid to surface visualization. Computer Graphics. July, 1983: 23-30.

[8] Blinn, J. F. and Newell, M. 1976. Texture and Reflection in Computer generated images. Communications of the ACM. October 1976: 542-457

[9] Sim, E. Face Modulation Options, (G.E. Technical Report PIR 2363-652-ES-242) April, 1981.

OTHER REFERENCES CONSULTED

Baker, W. General Electric Company, Daytona Beach, Florida, Personal Interview, October 15, 1986.

Gehret, T. General Electric Company, Daytona Beach, Florida, Personal Interview, November 7, 1986.

Gilbert, P. Software Design and Development 1983. Chicago, IL: Science Research Associates, Inc. 1983.

Hancock, L. General Electric Company, Daytona Beach, Florida, Personal Interview, October 29, 1986.

Lott, W., Computer Image Generator Lecture and Personal Interview, December, 1984.

Vaguerizo, J. General Electric Company, Daytona Beach, Florida, Personal Interview, November 5, 1986.

Wah, Y. L. Interactive Texture User's Manual,(G.E. Technical Report DOC 63A141794) August 1986