REALTIME IMAGE GENERATION BY SYNCHRONIZED TEXTURE CASTING

Raymond V. Sutton
B.S., University of Portland, 1964
M.S., Air Force Institute of Technology, 1966

A thesis submitted to the faculty
of the Oregon Graduate Institute
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science and Engineering

September, 1990

The thesis "Realtime Image Generation By Synchronized Ray Casting Through Textures" by Raymond V. Sutton has been examined and approved by the following Examination Committee:

Robert G. Babb II
Associate Professor
Thesis Research Advisor

Wm. Leler
Adjunct Professor

Jonathan Walpole
Assistant Professor

TABLE OF CONTENTS

ABSTRACT

REALTIME IMAGE GENERATION BY SYNCHRONIZED RAY CASTING THROUGH TEXTURES

Raymond V. Sutton, M.S.
Oregon Graduate Institute, 1990

Supervising Professor: Robert G. Babb II

Realtime computer generated imagery is based on filling polygons using highly pipelined processors; thousands of polygon/triangle processors operating in parallel have been considered when more detail is desired. For some applications such as automobile simulators, it may be possible to reduce the hardware and cost while providing realistic detail by changing the displayed primitive from a polygon to a flat, full scale texture map. This paper describes a system for rendering texture maps in parallel and in synchronization with a raster scan. Ray casting is done by chips dedicated to textures in their own memory, the closest opaque is chosen via arbitration on a bus, and the closest opaque and closer shadows are rendered via the bus. The results of a software simulation and hardware analysis of the system are presented, with emphasis on the design choices necessary to make the scheme practical.

1. Introduction

Aircraft simulators based on computer generated imagery have proven to be a cost effective way to teach flying to military and commercial pilots[9, chapter 11], especially in dangerous conditions. Similar devices might supplement current driver's education courses by demonstrating driving in dangerous situations and weather, and by allowing practice without the cost of an instructor. However aircraft type simulators are unsuitable for simulating cars due to their high cost relative to cars. The major hardware component of an automobile simulator will be its image generators. This paper describes a low cost, realtime image generator suitable for driver's education.

The principal requirement of an image generator for driver's education is a realtime view of roads, signs and other vehicles through a car window with one rearview mirror. A range of simulated environmental conditions is desired so

Figure 1. Image Requirements for Driver's Education

Required Feature	Design Implication
Essential object surfaces	
Diffuse (rather than specular)	Ray casting (rather than ray tracing)
Signal lights	Selflit and switchable surface types
Signs readable while moving	Antialiased texture maps
Textured roads to judge speed	Antialiased texture maps
Essential object shapes	
Buildings, cars, roads	Objects have nearly flat sides with irregular edges
Environment	
Fog (instead of rain, snow)	Objects obscured with distance Number of visible objects can be limited
Diffuse overhead ambient	Most shadows built into surface colors Moving vehicles need attached shadows
Extra views	
A rearview mirror per window	Mirror surface type, virtual eyepoint
Ice reflection	An inverted world as seen through ice *holes* in the street
Speed	
Relative speed of cars 120 mph	Rendering rate = monitor refresh rate

the training can include the problems of dusk and bad weather that are so hard to teach without a simulator. The detailed requirements for such an image system are outlined in Figure 1, along with their implications for the image generator design. Two implications have a major impact on the design. The first is the appropriateness of planar texture maps with *clear* elements for modeling the essential objects in the training model. It is so approriate to the model that all objects are modeled as texture arrays. The second major implication is the speed at which objects must be modeled. If the model must be rendered every time it is written to the monitor screen, there is little use for a refresh buffer and pixel color generation must occur on the average at the pixel clock rate. Since the pixel clock rate of monitors is approximately the clock rate of Very Large Scale Integration (VLSI) CMOS logic, the textures must be rendered in parallel. The requirements review suggests that the model world for automobile simulation might be closer to the theater, with its painted flats and controlled lighting, than the shaded polygons of common graphics systems. In such a 'theatrical' model, image generation would be accomplished by ray casting in parallel through a model constructed with planar, full scale texture arrays with transparent elements. The analogy to theatrical scenery suggested the common name for the texture arrays — *flats.*

In the Synchronized Texture Casting architecture, each flat array is stored in the texture memory of an "application specific" integrated circuit (ASIC) responsible for its display. For each pixel in the raster scan, each chip selects the element cast to in its texture. The independent images for each texture are then composited by a pipeline distributed across the chips. In the pipeline, the closest opaque element is chosen by a proximity arbitration between chips, then the chosen chip renders its element while closer shading pixels modify the image. Such an architecture provides an inherently realtime display of a number of surfaces. The number of surfaces is limited by the hardware, but it may be increased by the addition of ASIC/memory chip pairs to the hardware. The surfaces can be as detailed as the texture and color resolution allow. The scheme's disadvantages include ray casting's limitations, the preprocessing required to convert graphic objects into texture arrays, and those objects that can't be reduced to a few planes. Ray casting was chosen as adequate for automobile simulation, that is general reflection, refraction, point light sources and their shadows are not required. The simplest and most accurate way to model the everyday world is to digitize photographs of its surfaces, so preprocessing is not a problem. The number of surfaces needed can be controlled with some 'theatrical' license in the choice of objects modeled.

The following sections describe the Synchronized Texture Casting scheme, a software simulation of its operation, the simulation results and the planned hardware implementation. The description is organized around the system integration techniques used and the design choices made. The conclusion discusses each of the design choices in turn.

2. Related Work

2.1. Driver's Simulators

Driver's education usually ends in sessions with one car plus one instructor per student, but this is too expensive for more of the course. Mechanical aids that allow one instructor to teach a whole class simultaneously come in several varieties[13].

(1) *Cockpit per driver and a shared film.* The feedback to control input is not from the scene but from the instructor if the input is sufficiently wrong.

(2) *Cockpit per driver with shadows of a miniture model cast to a screen.* The feedback to control input is immediate but the model is static and not clear.

(3) *One of several cars on a test track with radio contact to an instructor.* The test track and need for visual contact with the instructor limit this aid to intersections and parking.

Despite their limitations, these teaching aids are effective, less expensive than one instructor and car per student, and liked by the students. They can be considered as part task teaching aids rather than complete simulators; they teach such skills as survelliance, braking or parking. None of them, nor even the usual one instructor and car session, cover hazardous conditions and weather. No complete, aircraft type simulators are in common use for driver's education.

2.2. Texture Maps

Current aircraft simulators are based on points and polygons, and usually use high speed, highly pipelined scan line techniques[9, chapter 4]. To achieve the detail required for signs in a driver training model, many small but complex polygons are required, which strains the processor speed. Texture mapping has been used as a way to simply render detailed objects since the early days of graphics[8]. To render a texture map, a raster position is mapped in perspective to an object position and then to a texture map location, where the location's color and often other characteristics are found. The object surfaces are often curved. The approach has not been converted to hardware, apparently because of the perspective divides needed at each pixel, but single numerator/denominator calculations have been implemented as chips[11].

Texture maps are not applicable to realtime scan line techniques. Scan line techniques gain their advantage by calculating groups of pixels at a time. With

textures, each pixel must be separately accessed from the texture arry; if it is *clear*, the next deeper texture must be tried. Thus at least one and often more clock cycles are necessary per pixel, so the average is greater than one clock per pixel. As a result, textures must be rendered in parallel when using dense forms of logic such as CMOS.

2.3. Parallel Rendering

There are two basic parallel image generation techniques. In the first, objects are assigned to processors and pixels are passed by the processors. Each processor is responsible for coloring pixels with its object's color if it is visible. Object parallel rendering has been implemented for the cases where the objects are polygons and triangles[1][4]. In the second parallel technique, pixels are assigned to processors and objects are passed by the processors[5]. In the pixel processor case, a quarter million processors are needed for a driver's education scene; in the triangle processor case, about a thousand processors are needed for active triangles. In both cases, the processors are simple, and the authors in the pixel and triangle cases suggest that a hundred and ten processors respectively can be accommodated per VLSI chip. In the best (triangle) case, this still requires a hundred chips per window of a simulator. Since both approaches use polygons as their primitive object, they also require a preprocessor capable of locating tens of thousands of polygons per raster frame, or tens of millions of floating point operations per second. These approaches are too expensive for automobile simulation.

3. Synchronized Texture Casting Architecture

The caster card planned for driver's education draws raster displays in real time by ray casting to flats in parallel and in synchronization with a raster scan. In the first major integration technique applied, a caster chip and its associated texture memory are responsible for rendering a flat. The display procedure is outlined in Figure 2 and the card block diagram of Figure 3. The interchip bus identified in the figures forms a 22 bit digital wired-or bus, which is the second major integration technique. During the horizontal and vertical synchronization portions of a raster scan, it is used to load 16 bit words to the caster chips and through them the *Texture SRAM*. During the active portion of the scan, it provides both proximity arbitration and output composition. Priority encode/decode hardware controls which chip on the bus uses it for input data during syncs and its output portion when active. The data transfer to the caster chip's control registers and texture memory is directed by the *loader state machine* of Figure 3 and occurs between main memory and a FIFO

Figure 2. Image Generation for One Raster Field

Procedure	**Unit/Bus Lines**
For each flat/viewpoint combination in the world model	
Transform flat to viewer/window frame	Main program
If flat in window and within visible range (through fog)	Caster driver
If flat not loaded in a caster	
Select an available caster	
Prepare display list to expand flat into memory	
If new or modified, write color lookup table	
Prepare display list for the flat in its section	
At start of vertical sync, load each caster's display list	Loader + casters
During remaining syncs, load and expand any new flat	Loader + casters
For each pixel in the raster scan, cast a ray to all flats	Parallel Casters
If the ray misses the flat, skip pixel	
If *mirror* color and enabled, drive *mirror* line	*mirror* line
Else if flat has wrong visibility, skip pixel	
If color number is not an enabled *tint/glass*,	
Participate in depth arbitration for closest opaque with final selection by priority encode/decode	*proximity* lines
If closest opaque, drive flat-in-caster and color number	*flat/color* lines
If enabled *tint/ glass* closer than opaque, drive outputs	*tint, glass* lines
For each pixel, combine color and environment values	Environment LUT + Color LUT

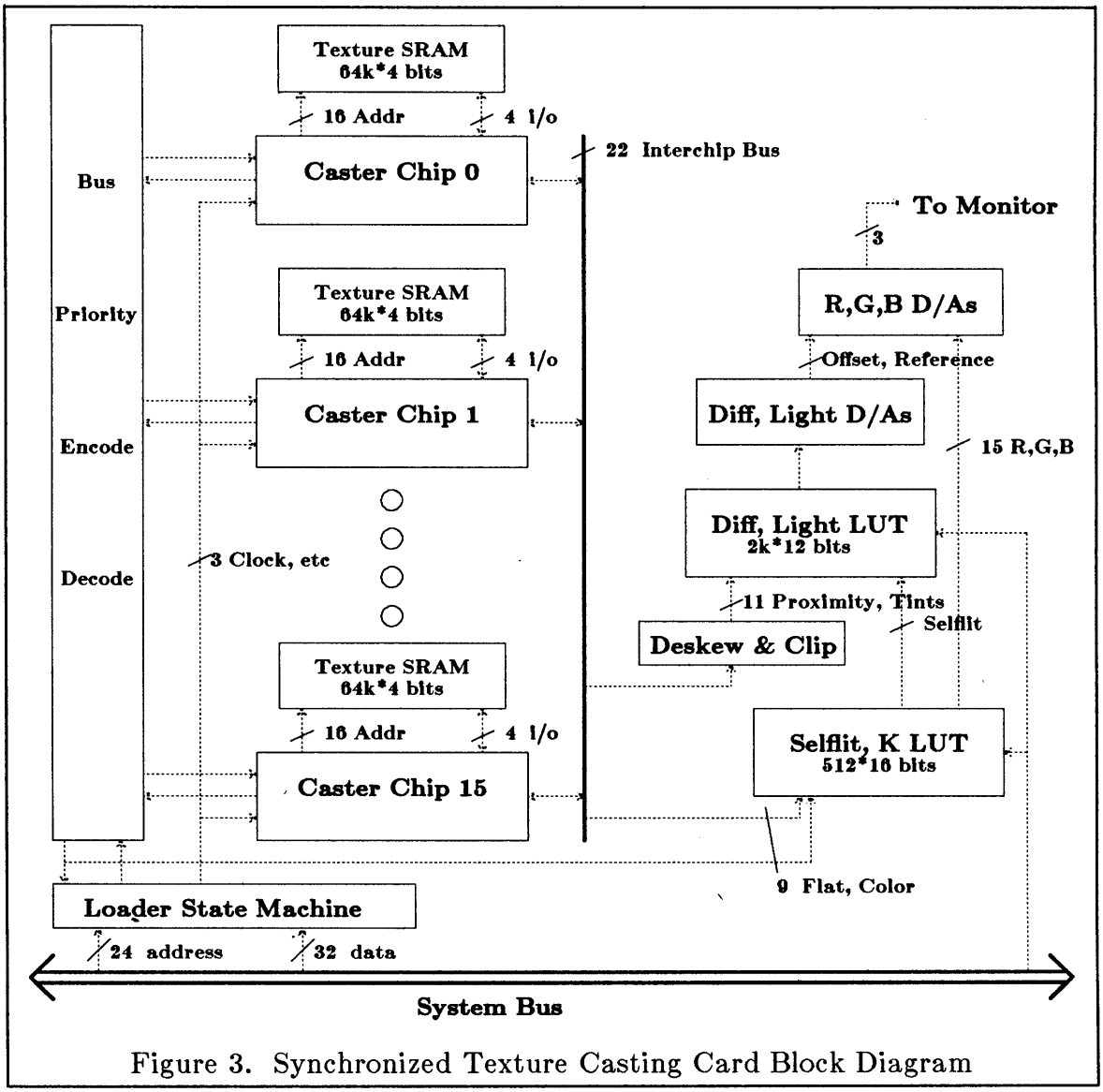

Figure 3. Synchronized Texture Casting Card Block Diagram

queue in each caster chip.

In the first major design choice, an image is limited to approximately 24 flats. In the second major design choice, each of sixteen chips handles up to two flats whose bounding boxes don't overlap horizontally. The nonoverlap requirement means that all chips can't be assigned two flats. The horizontal resolution is set by the caster chip at 640 pixels per line (512 pixels in the software simulation). The pixel rate and number of lines are set by the card as appropriate for NTSC or VGA monitors. The pixel periods of 80 and 40ns for these monitors are major factors in both the caster chip design and the cost of their texture memory. However, the basic caster functions can be heavily pipelined. Basically eight additions and a texture memory access per pixel by

each casting engine are needed to orient and display a window's view of a world of flats. Details of the ray casting, distance arbitration and pixel rendering calculations are described below in turn.

3.1. Ray Casting

The third major design choice is ray casting to full-scale, planar texture maps, in which case the texture to object transformation is unity and only the object to screen transform is needed. Considering that most objects important to the training model can be based on flat surfaces, the planar design choice seems reasonable. Assume the simulation program provides the positions of a flat's corners in an eye coordinate system (XYZ) oriented to the window's surface, which is at a distance (D) from the eye as shown in Figure 4. As each pixel occurs in the raster scan of the window, a ray is cast through the pixel to the flat. To calculate the texture cell indexes (u and v) and the flat depth (Z) hit by the ray as a function of X and Y, they are fit to planar equations based on their values at three of the flat's corners. The coefficients of the plane equations are selected by assigning indexes of 0 and 1 to opposite flat edges, so that a flat is present when both indexes are between these values and *clear* otherwise.

$$Z=Cz+Czx\cdot X+Czy\cdot Y; \quad u=Cu+Cux\cdot X+Cuy\cdot Y; \quad v=Cv+Cvx\cdot X+Cvy\cdot Y;$$

X and Y can then be replaced by the window's pixel coordinates (x, y) using the perspective equation[6].

$$x=X\cdot D/Z; \quad y=Y\cdot D/Z;$$

The resulting equation for Z can be solved to find Z as a function of x and y, and the result substituted in the u,v equations. In the caster chip, the flat coordinates for the texture cell and the flat proximity [7] can then be calculated

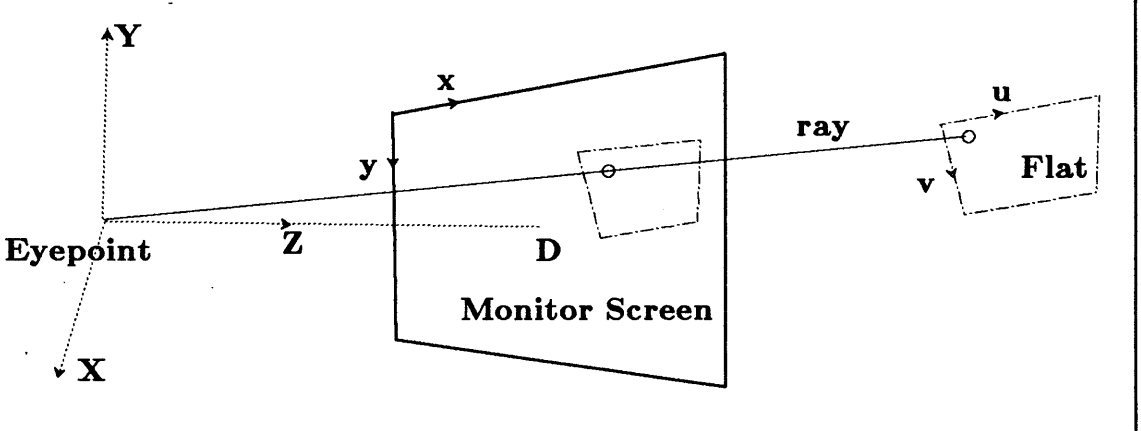

Figure 4. Ray Casting Coordinate Systems

with the equations:

$$p \equiv \frac{1}{Z} = cp + cpx \cdot x + cpy \cdot y; \quad u = \frac{cu + cux \cdot x + cuy \cdot y}{p}; \quad v = \frac{cv + cvx \cdot x + cvy \cdot y}{p}$$

Since the raster scan uses these calculations in sequences of x and then y, the multiplications can be replaced with repeated addition, and only the additions that depend on x are required for every pixel.

Most three dimensional graphics systems use floating point arithmetic for accuracy over a large range of values. The fourth major design choice is to use a form of block floating point in the calculations. The coefficients are all shifted as a group so that integer overflow is just avoided when the numerator and denominator are calculated at the bounding box extremes. The quotients are unaffected by the group shift. The only place the block exponent is used explicitly is when proximity p is shifted in hardware before proximity arbitration.

A direct implementation of these equations would also require two divisions per pixel per flat, which would require half of the chip's area. Instead, in the fifth major design choice, serial divisions calculate u and v at the end points of short horizontal segments, and p, u and v are then linearly interpolated between these points. Interpolation should be done in power of 2 segment lengths because a division by the segment length is necessary in the conversion from change per segment to change per pixel. Because greater than eight bit precision is required in the division (8 bits for u plus one bit of overrange test and 6 fraction bits to provide accurate interpolation) and each bit takes a clock cycle in serial division, interpolation segments of 16 pixels would be convenient. Unfortunately, 16 pixel segments converted straight edges in flats to curves with cusps at segment boundaries, which appear as scallops. Therefore 8 pixel interpolation segments are required, and the divisions must be done in two sections. Minor scalloped edges are still visible with 8 pixel interpolation on a few tilted flats, as will be shown in Figure 17.

The Caster Chip Front End block diagram shown in Figure 5 summarizes the calculation pipeline. Four groups of eight clocks make up the pipeline. The first clock group uses one adder to do all multiplications by repeated addition and transfer the results to the remaining data path elements. The second and third clock groups use four adders for the u,v serial division stages, while the fourth group uses three adders for interpolation. Two additional adders in a single clock stage handle antialiasing, as will now be described.

3.2. Antialiasing

The casting procedure so far makes no provision for spatial filtering of the texture arrays, which can lead to aliasing[8]. The problem occurs at two levels, in building the texture arrays and in casting them to the screen. The first level

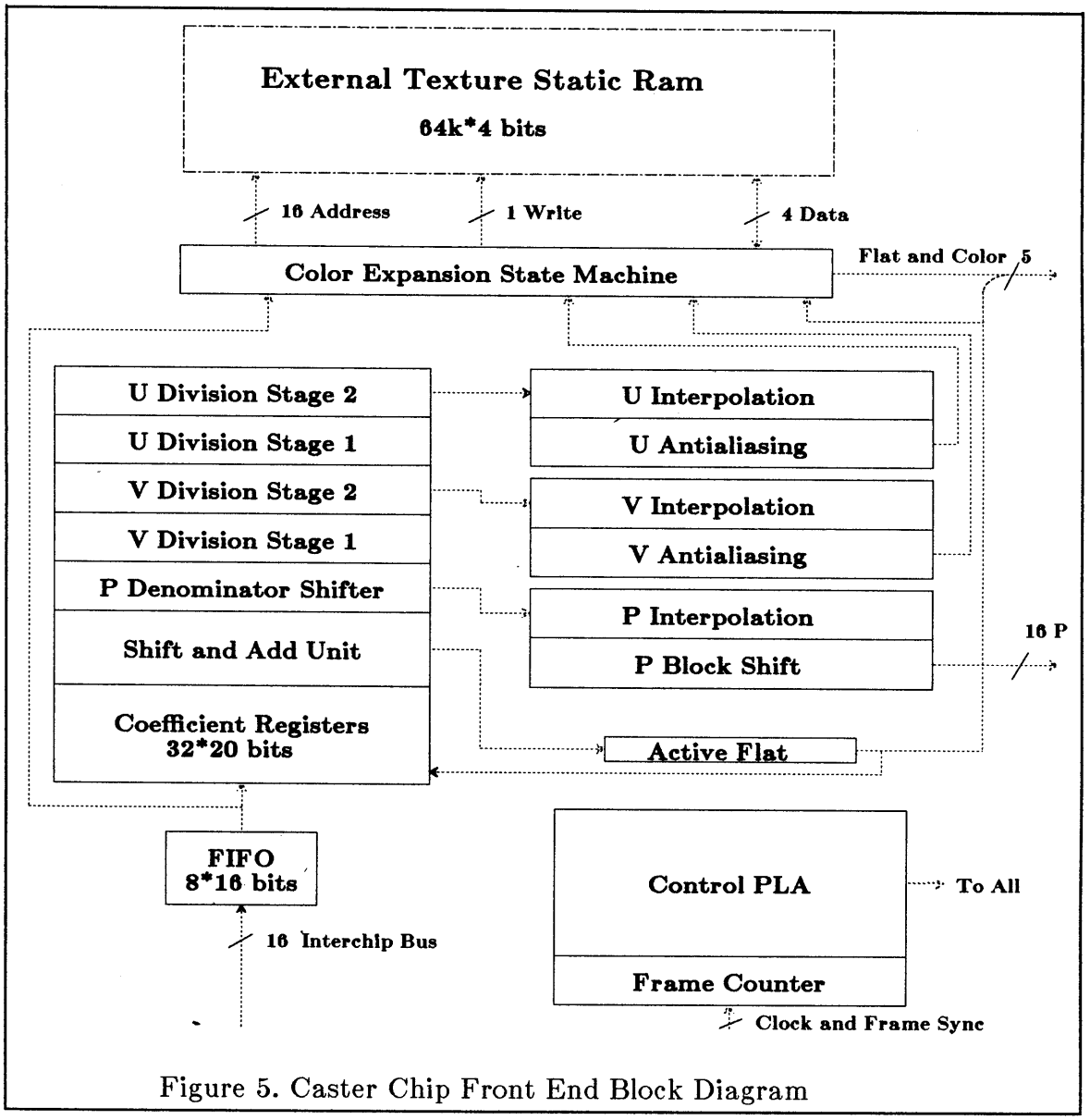

Figure 5. Caster Chip Front End Block Diagram

occurs once during flat construction, and is discussed in the tools section. Antialiasing at the casting level can be simplified by using another old technique — provide multiple texture arrays for one surface with different resolutions[12]. In the literature, several different resolutions of the same texture are often combined in the same file. In this case, casting to low resolution flats would still take a full flat's hardware, which is the major cost factor in this architecture. To reduce the number of low resolution (distant) arrays, several might be combined into one flat in the automobile case. Precombining distant flats should be effective in automobile simulation because the flats combined can be chosen to take advantage of viewing angle

restrictions due to the fact that vehicles are limited to roads.

Probably no more than one precombined set of flats will be required because another antialiasing technique is available, although it has a limited resolution range. In the sixth major design choice, filtering is included as an average by the eye over several fields of a jittered regular sampling grid[3]. By experiment with the software simulation, jitter was chosen to be the maximum of ±¼ of a texture element, ±¼ of a screen element and the fourth power of the ratio of the flat's average depth to the visual range. The first two terms cover the cases where the texture element is larger or smaller than a screen pixel, respectively. Since self-lit pixels are visible to 1.5 times the visible range, the third term provides a maximum corona[10] of ±5 screen pixels around lights.

The use of jitter for texture coordinates would ordinarily require multiplication of jitter size by a weighted random number. Bessel weighting is appropriate for corona[10], where the jitter shape will be most obvious. Because the multiplier is random, the product can be approximated by a bitwise *and*. Other operations at the line and pixel level vary the multiplicand to approximate the Bessel shape; the first is done by the basic adders during line setup calculations and the second uses a bitwise *or* at every pixel to weight the multiplicand to a number between ½ and ¾. The continued adjusting of the multiplicand also masks any pattern in the jitter due to the bitwise *and* product. The operations are (**C** syntax):

```
float weight[4]={1, 15/16, 7/8, 3/4};

line_jitter=jitter_u * weight[prg_u()&3];

pixel_jitter=(prg_u & 1) ? line_jitter : (line_jitter>>1 | line_jitter>>2);

pixel_delta=pixel_jitter & prg_u;

u'=u + ((prg_u() & 2) ? pixel_delta : ~pixel_delta);
```

3.3. Proximity Arbitration

The closest opaque texture cell is located by comparing p between those caster chips that have cast to an opaque color in their flat. As part of the second major integration, this comparison is done on the interchip bus rather than in separate hardware. The comparison also provides a measure of distance from the viewpoint, which is needed to render fog. The arbitration is done serially, most significant bit first, on the wired-or (precharged) bus. The arbitration is effectively parallel because the color rendering is delayed in each chip by the same amount.

A mirror signal is processed before the proximity arbitration; any flat with an enabled mirror element in the ray discharges the mirror line and causes the

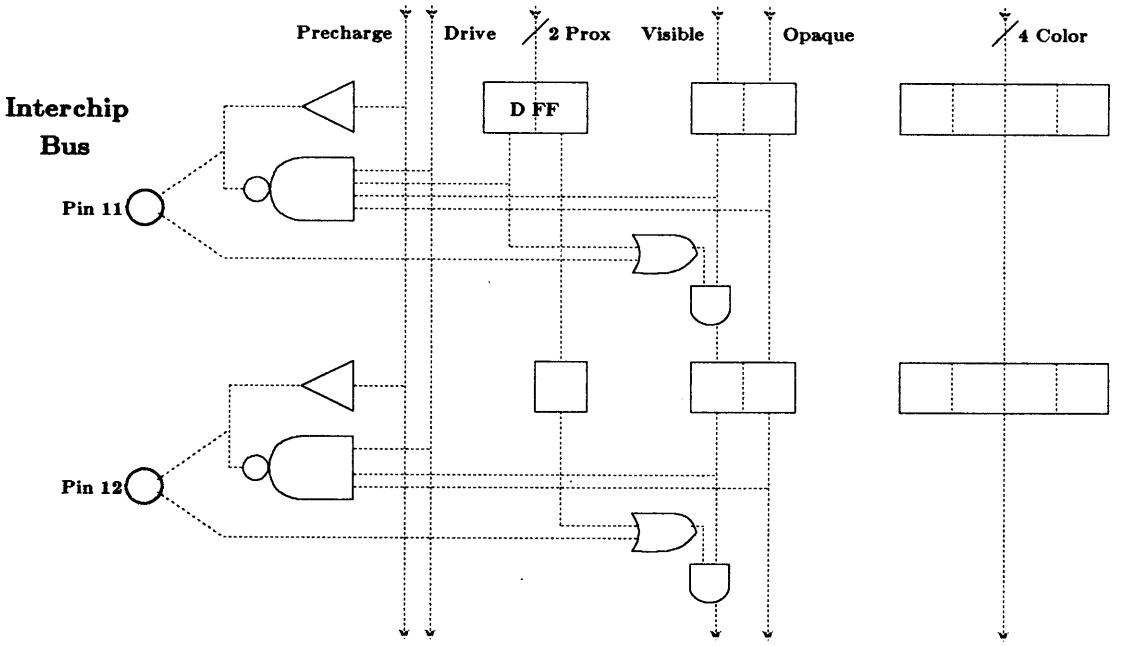

Figure 6. Two Stages of the Caster Chip Arbitration Pipeline

arbitration to be done between those flats flagged as visible in the reflected space rather than in the direct space. This technique does not allow the mirror to be hidden by closer opaques, but avoids the need for both a reflected and direct view proximity arbitration and satisfies the automobile simulation need for one close rearview mirror per window.

The 16 bit proximities in the caster chips could be compared directly using 16 bus pins, but this would be excessive since most of proximity's resolution is concentrated at high proximity/short distances. A more even distribution of distance resolution can be achieved by switching to a floating point format with two exponent bits before comparison, which is an additional function of the first two pipe stages. In this way the number of bus pins needed for proximity comparison is reduced to 12.

The proximity arbitration itself is done a bit at a time, starting with the most significant bit, searching for the largest proximity (closest distance). Referring to the logic of the last two arbitration stages in Figure 6, those casters with an opaque, visible cell in the higher half of the proximity space being processed by a particular bus line discharge the line, and any casters with visible cells in the lower halfspace when the wired-or line is discharged are identified as hidden and drop out of the arbitration. The two pipe stages following proximity arbitration contain the priority encode/decode hardware used for final caster selection.

An ice view is expected to be necessary for training since ice is usually detected visually by image motion in it relative to the street. Ice reflections could be handled as another mirror type, but the fact that ice can be hidden by closer flats would necessitate parallel direct and ice proximity arbitration, and then use of the ice flats if the closest direct surface was ice type. Fortunately a hardware solution is not necessary since components of the direct and ice reflected view are not superimposed in the model. *Clear* or *tinted* holes are left in the ground surface of the model, so that a copy of the direct model inverted about the ice plane can be seen when proximity arbitration doesn't indicate a closer ground surface. To 'roughen' the ice, jitter is increased by 1 screen pixel per 2.5 meters that ice reflected flats are behind the ice, with a minimum increase of 1.6 screen pixels.

3.4. The Texture Map

In the seventh major design choice, flats are texture arrays of four bit color numbers with dimensions of 256 by 128, so that two arrays will fit in a 64k by 4 static RAM paired with each casting chip. This design choice sets limits on the texture array aspect ratio, element resolution and color resolution. The physical aspect ratios of the example flats in Figure 7 support the choice of texture element aspect ratio; the mean aspect ratio is 2.5:1, with 2:1 the closest binary ratio. Note that streets can have higher aspect ratios which are really limited only by surface topology. The suitability of the 32 thousand element arrays will be shown by example later. A lookup table to convert color number to RGB intensities is necessary so that signal lights can be switched on or off without modifying the texture array, but it also means only small color numbers need be stored in the texture array. In the example flats in Figure 7, all fifteen assignable colors per flat were used only to add detail inside houses as seen through the windows.

Image data can be very large; the fifty flats that might be used in a local area would require two thirds of a megabyte uncompressed. A training model consisting of several hundred areas would exceed optical disk capacity. Existing image compression chips could improve this situation dramatically. However, another problem occurs when texture maps are moved from processor memory to a caster chip's texture memory. For minimum chip pinout, several new flats would be loaded to the caster chip over the interconnect bus and then to the texture map during one scan synchronization period, which is too much with uncompressed flats. Therefore, flats are compressed by an algorithm for which decompression requires little logic for expansion on the caster chip.

Several compression schemes were considered to take advantage of the high spacial coherence of flats. The first scheme attempted encoded single color runs as a color number and a short length or a long extension of the last color. This scheme was counterproductive when halftone techniques from printing technology were used to modulate between colors (i.e. trade element resolution

Figure 7. Characteristics of Binary Flats in Software Simulations

Flat File	Flat Bytes	Comp Ratio	Physical Size(cm)	Aspect Ratio	Colors Used
ball.F	969	18.18	30*24	1.25	3
bush.F	3527	4.74	300*200	1.50	4
car.right.F	2589	6.50	396*128	3.09	13
car.shadow.F	3861	4.12	520*300	1.73	7
hood.F	1580	10.84	140*114	1.23	3
front1.F	3377	4.95	1000*350	2.86	15
front2.F	3592	4.65	1000*350	2.86	15
garage01.F	3617	4.62	1000*304	3.29	12
lawn12.F	3759	4.44	2690*1383	1.95	12
mirror.F	806	22.20	18*6	3.00	3
roof1.F	3757	4.44	1000*400	2.50	3
roof2.F	1744	9.77	1000*400	2.50	5
sign.185.F	2040	8.31	400*150	2.67	4
signal.F	2232	7.34	600*145	4.13	11
sig.shadow.F	1675	10.20	600*145	4.13	7
street.light.F	2088	8.11	570*50	11.40	4
steps.F	1296	13.34	140*200	1.43	2
street.e.F	2993	5.60	2700*1358	2.45	7
street.F	3283	5.10	3300*1250	2.64	14
street.ne.F	3826	4.36	2690*1383	1.95	6
tree.F	3265	5.12	550*400	1.37	5
tween12.F	3052	5.49	2500*1120	2.23	9

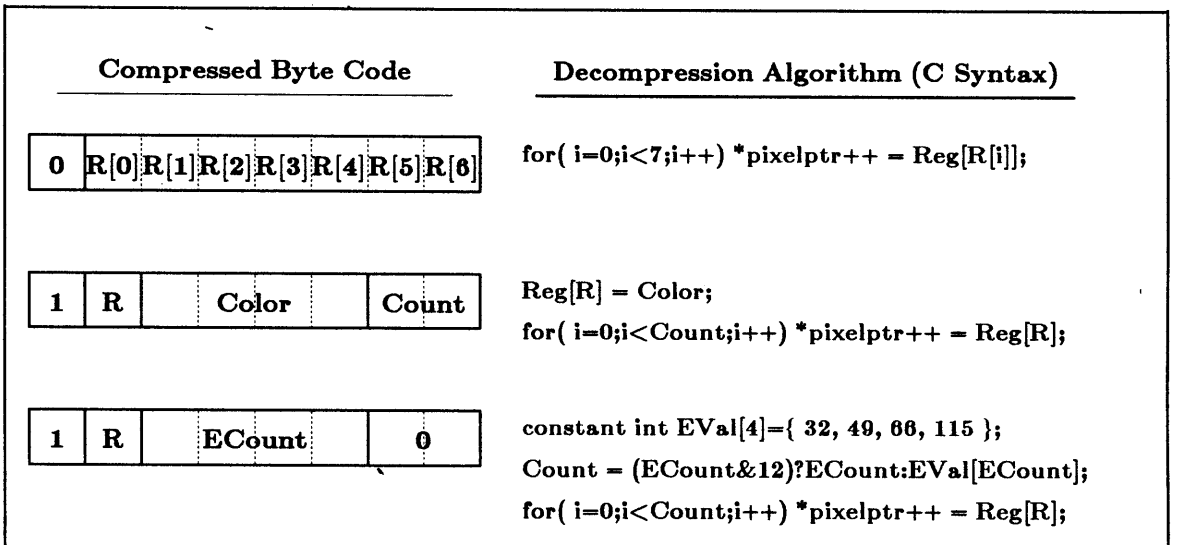

Figure 8. Two Color Flat Compression Coding

for color resolution). In the two color compression used in the simulations, horizontally repeated colors or sequences of color pairs are encoded into bytes with the help of two registers[2] as shown in Figure 8. The scheme also allows signs to be created on the fly because a compressed flat consisting essentially of codes with seven register indexes can be considered a bitmap and can be modified with character bitmaps. The compression ratio of the flats used in the examples is between 4 and 6½, as shown in Figure 7, with simple flats going higher. Further compression could make use of vertical coherence, but would sometimes require both a read and a write to the texture memory for one pixel, and was judged to take too much decompression time as well as logic.

3.5. Rendering Calculations

The requirements of Figure 1 include the need to render textures with ambient and self-lit diffuse elements plus transparent elements in fog. Thus a texture cell's color number usually specifies one of 15 opaque surfaces or a *clear* (Color 0) area. Shadows and tinted car windows would add to the reality and depth perception of moving vehicles. Since there are few transparent types needed, each can be assigned a specific color number and a type enable bit. Problems could arise if transparencies overlap from different flats, since there is no easy way to sum their effects on a digital bus. A shift register connection of the chips or an analog current sum portion of the bus would have solved the problem, but at the cost of a larger chip for the extra pins or D/A converters. Fortunately car shadows are placed just over the surface a car rides on, leaving little chance for two shadows to overlap. Windows can overlap, so only one wired-or bit is used for them, for the effect of sum and clip. Interactions between these two kinds of tints are avoided by combining each on separate lines of the bus.

The basic rendering equation[6], with self-lit, shading and environmental[10] affects, is shown first in Figure 9. Distance from the viewpoint for calculating fog effects is approximated by proximity, which is related to the model Z dimension rather than true distance. Because of the limited dynamic range of light from CRT monitors, self-lit surfaces representing signal lights can not have their true relative brightness without fading the rest of the scene to insignificance. To make maximum use of the monitor's available range, the basic equation is modified as shown in the second equation so that self-lit surfaces attenuated by fog can still drive the monitor to a true color maximum output. To allow color changes in realtime (e.g. turning a signal light surface on and off), the equation is implemented as two lookup tables (LUTs) with analog and digital connection as shown in Figure 3 and the last equation of Figure 9. Global values such as ambient and visual range are written into the Environment LUT.

Figure 9. Rendering Math and Its Look-Up Table Approximation

$$i_{R,G,B} = A\left(1-e^{-1/p}\right) + \left((s)?\ 1 : A\ \frac{10-t}{10}\right) e^{-1/p}\ \frac{2-g}{2} K_{R,G,B}$$

With dynamic range extension, let: $F = A\left(1-e^{-1/p}\right)$

$$i_{R,G,B} = F + \left((s)?MIN\left(10\ e^{-1/p},\ 1-F\right) : A\frac{10-t}{10}e^{-1/p}\right)\frac{2-g}{2} K_{R,G,B}$$

$$= D[p] + L[s][p][t][g]\ K_{R,G,B}$$

where:

$i_{R,G,B}$ are the cast ray's analog RGB intensities
A is the ambient's fraction of a monitor's maximum output
p is the proximity of *closest* opaque flat, scaled to ¼ at visual range
n is the closest flat's number [0→31]
c is the closest opaque color's number [1→15]
$s[n][c]$ is the closest opaque pixel's self-lit flag (from Color LUT)
t is tint (color numbers 0→7 when tints enabled)
g is glass (color number 14 when glass enabled)
$K_{R,G,B}[n][c]$ are ambient-lit opaque paint's reflection coefficients or self-lit paint's intensity (from Color LUT)
D,L make up the Environment LUT

3.6. Hardware Estimates

A casting system able to render a maximum of 32 flats on a simulated window in realtime is expected to require approximately 60 CMOS VLSI chips, and thus would fit on a single printed circuit board. Half of the chips, the caster chip/memory pairs, can be tightly packed such as on a memory card. In fact, the chip pair might be mounted in a single inline package with a pinout of the 22 bit interchip bus and a few power and support pins. The caster chip's size depends on the functions described and their accuracy. The array index numerators, denominators and division were found to require 20 bit precision, while the index interpolation required 16 bits. Based on an estimate of the number of logic components and an estimate of the number of gates in each, the caster chip is expected to require 5,000 to 7,000 gates and 48 pins. Thus it is a small to medium ASIC gate array.

The image generation computer needed to drive a caster card should be in the single board category because the computer will transform three corners of perhaps 100 potentially visible flats in a typical portion of a driver's training model at 30 or 60 frames/second, or 400 thousand floating point operations per second. Image generation storage will depend on the total amount of

imfomation stored. Assuming 4 kbyes for an average flat plus its control information and fifty flats per 200 feet of city street allows over 100 miles of street in a driver's education model that fits on an optical disk. This modeled distance would support a square mile city plus 50 miles of surrounding rural roads and freeway. If data for each 200 feet of street were contained in a 200k byte file, a Mbyte of memory might be needed to store potentially active files. At 60 mph, a new file would be loaded to memory every couple of seconds and a new flat to texture memory on the average every couple of frames. Because it would take about half a second to access and load a 200k byte file from optical disk, each file's need would have to be anticipated in software, perhaps by projecting the vehicle position forward several seconds.

4. Synchronized Texture Casting Software Simulation

Most opaque flats will be based on photographs of the sides of real objects for realism and to avoid manually entering impossible amounts of data. The exceptions are textual objects such as traffic signs, where the desired text may not be available as a picture. The texture arrays built by either technique are then placed in a model and a simulated image of the model created as the Synchronized Texture Casting hardware would. Only a sign editor to build flat files and a model editor/software simulator have been built; a photograph editor and realtime hardware were not built. The picture type flats built with the sign editor are not as varied or detailed as they might be because detail was manually entered. For driver's training, for instance, a flat of a child on a skateboard would be an appropriate problem but very difficult to build with the sign editor.

4.1. Software Simulation Hardware

To support this work, a Motorola VMEsystem 1121 workstation with 5 megabytes of memory running Unix System V.2.2 was used. A Matrox VIP-1024 image card with eight bit planes was added for the simulated vehicle window. For the animation, a Panasonic AG-1960 VCR recorded the Matrox card's signal with a Futurevideo EC1000 controller, Sony GBM-2000 monitor and Truevision Vidi/o S-Video converter.

4.2. The Sign Editor: signed

Texture maps of signs are built using an interactive graphics editor that operates on text files describing lines, arcs, outlines and text strings. The program displays the objects and reads the display to build a binary flat. In addition to the functions of a two dimensional image editor, the editor provides aspect ratio correction, flat compression and color definition tailored to the rendering system. A simple paint system at the flat resolution level was used to touchup errors in the flats due to aspect ratio correction and the character bitmaps used. To put as little strain on the realtime antialiasing system as possible where detail is important (e.g. signs), characters should use three color levels instead of just foreground and background. This was manually accomplished with the touchup system. Appendix 1 contains manual pages and an example object description file for the sign editor.

4.3. The Model Editor: moded

The model editor supports the creation of display scripts and shows the resulting images. All of the casting, day-to-dusk rendering and antialiasing

Figure 10. Caster Chip Registers

```
struct CREG {
   unsigned char top_line;     /* top of rendered part of screen */
   unsigned char fp_scale;     /* block floating point exponent;
                                  bit 7 = load flat's texture memory */
   unsigned char bottom_line;  /* bottom of rendered part of screen */
   enum flat_flags {  in_Ice_world=128, in_Reflected_world=64,
         self_lit=32, opaque=16, ice=8, mirror=4, glass=2, tint=1 };
    /* flat coordinate and proximity coefficients; bases divided by 16 */
   short cp, cu, cv, cpx, cux, cvx, cpy, cuy, cvy;
   unsigned short prg_initialize, jitter_u, jitter_v;
     } creg[2];              /* two flats per chip */
```

techniques described for the caster chip are included. The program is broken into ChipDriver, Chip and LUTDriver functions as they might be in practice. The **C** structure that forms the interface between the ChipDriver and Chip functions is shown in Figure 10, which could well represent the caster chip register definition. Because of the limitations of the eight bit planes of the display hardware used, a maximum of 16 unique flats without tints or 8 unique flats with only one level of tint are simulated. Appendix 2 contains manual pages and example ASCII files for the model editor.

A first major design choice of the Synchronized Texture Casting architecture is the limited number of flats in a particular hardware configuration. The following scene characteristics influence the severity of the limit. The techniques described for each characteristic are built into the model editor to reduce the number of flats where possible.

(1) *The type of objects in the scene.* Houses and office buildings can be made in a straightforward way. Trees might appear difficult but, with some theatrical license, they are considered cylindrically symmetric about the vertical, and the flat with their image is rotated to face the viewer. Trucks and vans are expected to take six to eight flats, but the rounded curves of most sports cars would require many more. Hence, there will be only flat-sided sports cars in the model world and only a couple vehicles at any time. The training value of a simulator is not expected to be compromised by the use of flat-sided objects.

(2) *The degree to which unnecessary flats are ignored.* Unnecessary flats include those offscreen, backfacing parts of solid objects, and reflected/ice flats not behind mirrors/holes. In *moded*, flats are rendered only if they appear in the appropriate clip box - the screen for directly visible flats and the bounding box of mirror/ice flats for reflected views. Flats are also dropped if they are not visible through fog; flats with

opaque colors are dropped at the visible range while flats with self-lit colors are dropped at 1.5 times the visible range. Flats labeled as unnecessary when backfacing, such as either the normal or reverse component of traffic signs, are not rendered.

(3) *The degree to which groups of distant flats are combined into single flats.* Combining is a necessary tool to avoid antialiasing as well as to reduce the number of flats. It was not attempted with the tools available.

4.4. Simulation Results

Figures 11 and 12 show typical driver's training views of a local area with over thirty six flats. To approximate temporal averaging by the eye, each image is a multiple exposure of three identical images except for different pseudorandom generator initialization. Figure 13's overhead viewpoint makes the construction by flats obvious; flats beyond those that could be handled by the software simulation are outlined. Figure 14 shows a scene with transparencies. Note that the flat images reflected by ice has been tinted at the edge of the ice and jittered an extra amount. The software simulation unfortunately limits the tint levels to one and the number of flats to eight unique opaque flats. In Figure 15, the same combination of a sign and signal lights is rendered at various distances relative to the visual range to show jitter/time averaging and rendering effects. In Figure 16, only one jittered frame is shown to show the degree of jitter. Note the corona around signal lights deep in fog due to strong antialiasing and extension of the monitor's dynamic range. Figure 17 shows the worst case example found of scalloping, on the far edge of the far roof. At that point, the maximum error of serial division/interpolation is just under one screen pixel.

Several animations were produced by recording frame pairs on a video recorder as the eyepoint was moved in the model world. In the first, the eyepoint was moved out of the driveway of the near right driveway of Figure 13, turned right and moved up to the signal. Figures 11 and 12 are from the middle and end of this sequence, but without a rearview mirror because of the 16 flat software simulation limit. The number of flats in this local area of 600 feet of street was 43. The maximum number of flats visible in any frame was 20 (some outlined), and these flats fit into a maximum of 12 chips when vertical overlap was considered. Note that another vehicle was not attempted. The second animation was a drive down the model street with a ball flying out into the street. Appropriate cautions were inserted after the ball as an instructor might do if the ball, and the child that might follow it, was not responded to properly.

4.5. Conclusions

The application of system integration techniques to the problem of ray casting through textures in realtime has resulted in the design of a single card

Figure 11. Driver's view 1 (visual range 100m, ambient 84% max)

Figure 12. Driver's view 2 (visual range 100m, ambient 84% max)

Figure 13. A distant view of a local area showing its construction. Flats beyond the software simulation limit are yellow outlines.

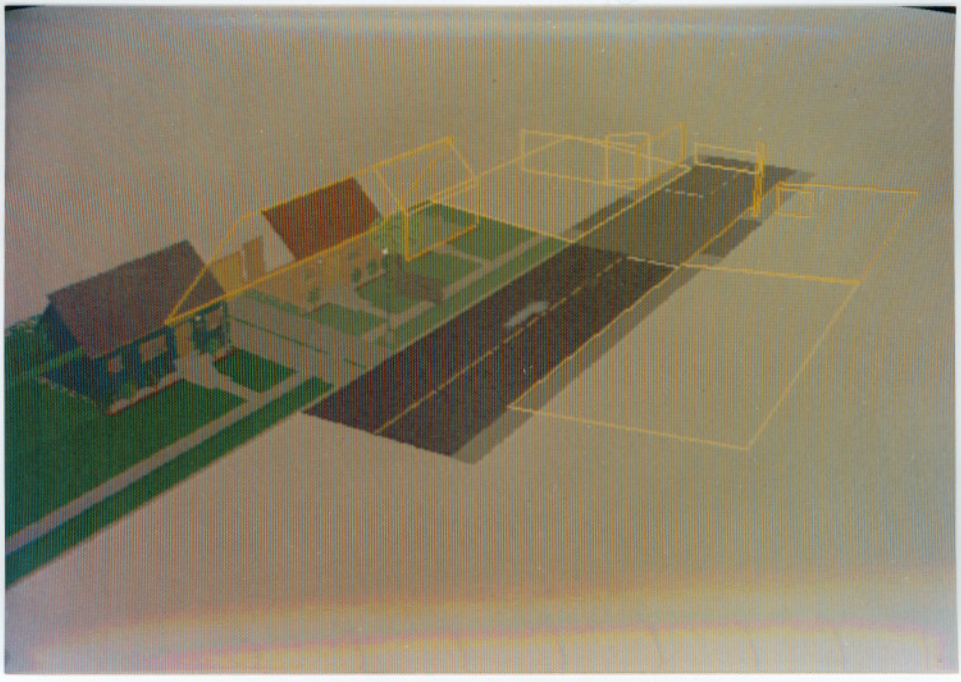

Figure 14. A driver's training model scene with tints and glass.

Figure 15. Jitter affects on a sign with signal lights at 0.05, 0.25, 0.50, 0.75, 0.90, 1.00, 1.10, 1.25 and 1.50 times the visual range.

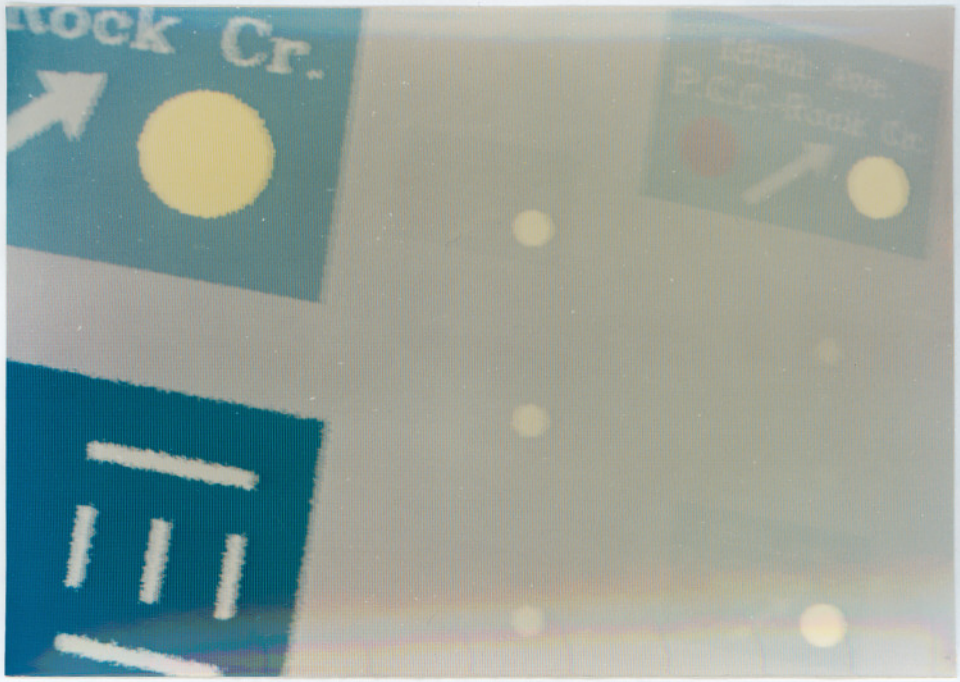

Figure 16. A repeat of Figure 15 without fog or time averaging.

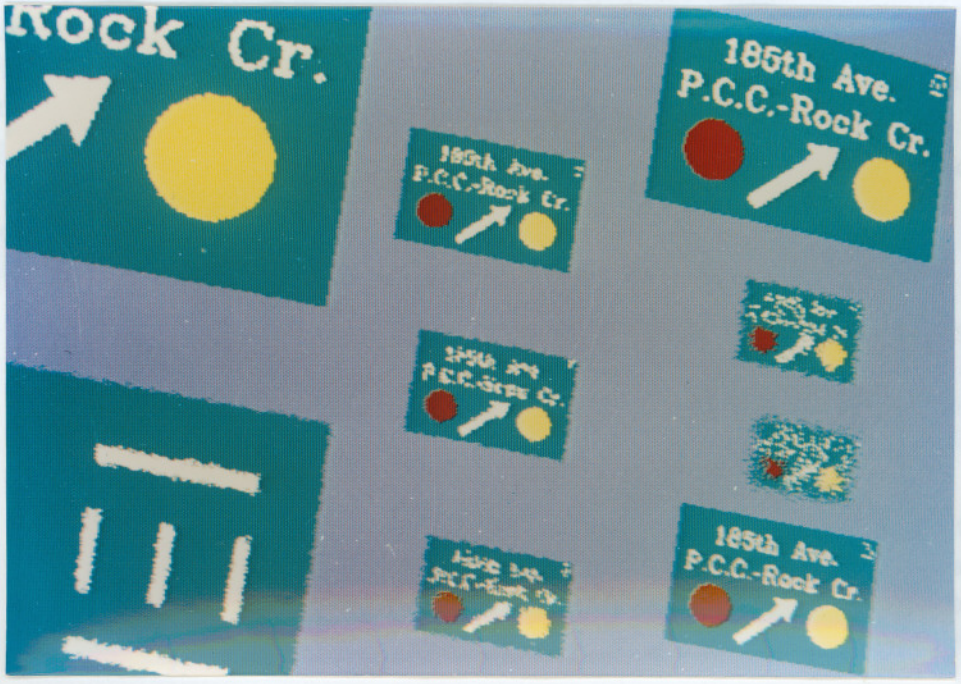

Figure 17. Scallops on the far roof due to 8 pixel interpolation.

for realtime image generation. The basic approach assigned each texture to a chip with private texture memory and connected the chips with a wired-or bus for distance arbitration and compositing. Other than requiring that all rendering be by ray casts to textures, the basic approach requires no performance compromises.

Many performance compromises occur in the implementation to make image generation cost effective for automobile simulation. The results of each of these design choices are described below in turn. The parameters used in the software simulation would make a good starting point for driver's education.

(1) The animations used a maximum of 20 flats/window in 12 chip pairs. This resulted in a satisfactory training model as shown in the animation, though it was without other vehicles. Twenty caster chips are recommended for a real implementation, which by extrapolation will display over 32 flats.

(2) The assignment of two flats that didn't overlap horizontally to each caster chip resulted in both sets of logic being used on average two thirds of the time. This result was not very satisfactory, but two flats/chip was required considering the size of flats relative to available static RAMs.

(3) Planar, full scale textures didn't appear to compromise the training model. Vehicle models have not been tested; not all vehicle types are expected to be easily modeled with planes.

(4) Block floating point operation was satisfactory when used with 20 bit indexing arithmetic.

(5) Serial division and interpolation were satisfactory when the length of the interpolation was limited to 8 pixels and 16 bits were used in the interpolation arithmetic.

(6) Temporal antialiasing via jittering of the sampling grid was accomplished with a small amount of additional hardware. As an added bonus, jittering provided corona around signal lights deep in fog. The animation was not accurate to a frame and so was not used to judge temporal antialiasing perpormance.

(7) The size of the private texture array memories of 256*128*4bits was satisfactory. Fifteen colors plus clear was not a constraint in the construction of the sample flats. If there are a few cases where more element and/or color resolution are needed, several flats could be used for the same surface.

The hardware design described and its software simulation demonstrate an effective approach to an inexpensive realtime image generator for driver's training. With available system integration techniques, realtime ray casting to texture arrays can be integrated on a chip and compositing can be done on a wired-or bus that connects them. The rendering can include the signs, signal lights, rearview mirrors, ice and fog essential to the training. The software load to drive the visual system is based on tens of textures rather than tens of thousands of polygons, which should allow display operation by single board computers. A complete driving day to night driving simulator would also need the point source lights discussed next. Thus Synchronized Texture Casting can be the basis of an affordable realtime display system for driver's training.

Additional applications include those where realistic flat surfaces are a major part of the model to be rendered. Because Synchronized Texture Casting can be combined with conventional scan line systems following the latter's frame buffer, flat surfaces do not have to be all of the model. Possible applications include architectural CAD and presentations, three dimensional cartoon backgrounds, ship and aircraft simulators and arcade games.

5. Future Work

The primary future work is to design the caster chip, have it built and modify the existing software to support it. Several enhancements to the design might also be considered.

(1) The number of caster chips, and hence the number of flats rendered, is limited by loading of the interchip bus. Multistage arbitration networks are possible but would double the pinout. A better alternative would be to place several caster/SRAM blocks on the same chip.

(2) The rendering approach described only covers daylight to dusk because it doesn't include light sources such as headlights and streetlights. The interchip bus, however, might also drive point source lighting chips. The lighting chip would use a mathematical description of a light cone plus raster position and proximity to calculate and add light to any closest opaque surface and fog that are in the light cone's volume. Combining the illumination and fog outputs of the multiple point source chips needed for night scenes is expected to require analog current sums on a bus connecting the point source chips. These sums would add to the environment D/A converter outputs.

(3) Slightly curved texture surfaces (e.g. car sides and lawns) would make the training model more realistic. Using quadratic equations for the flat index's numerator and denominator might add two adders and nine coefficient registers to the caster logic. The texture map coordinates could in general be mapped to a curved surface, but the mapping can't be inverted to provide texture coordinates as a function of screen position. This can be demonstrated by noting that a ray could strike a curved flat at two points, providing two sets of texture map coordinates. Partial solutions can be found to the inversion problem. When necessary, the surface could be cut at ray tangent lines and rendered as two partial surfaces fit to quadratic equations.

6. References

(1) Abram, G., Fuchs, H., "VLSI-Architectures for Computer Graphics", Advances in Computer Graphics I, pages 189-204 (1986 EUROGRAPHICS).

(2) Campbell, G., Two bit/Pixel Full Color Encoding, Computer Graphics, vol 20, No. 4, p 215, Aug 1986.

(3) Cook, R., "Stochastic Sampling in Computer Graphics", ACM Transactions on Graphics, Vol 5, No 1, Jan 1986.

(4) Deering, M., Winner, S., Schediway, B., Duffy, C., Hunt, N., "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics", Proc. SIGGRAPH '88, Volume 22, No. 4, August, 1988.

(5) Eyles, J., Austin, J., Fuchs, H., Greer, T., Poulton, J., Advanced Computer Graphics Hardware II, Record of Second Eurographics Workshop, 1987.

(6) Foley, J., Van Dam, A., "Fundamentals of Computer Graphics", Addison-Wesley Publishing Co, Reading, Mass, 1984.

(7) Grimsdale, R., "Techniques for Real-Time Image Generation", Parallel Processing for Computer Vision and Display Conference, University of Leeds, Jan 1988.

(8) Heckbert, P., Survey of Texture Mapping, Computer Graphics and Applications, Nov 1986.

(9) Schachter, B., "Computer Image Generation", John Wiley and Sons, New York, 1983.

(10) R. Tricker, "Introduction to Meteorological Optics", American Elsevier Publishing Co., New York.

(11) TMC2302 Data Sheet, CMOS Advanced Image Manipulation Engine, TRW LSI Products, La Jolla, CA Nov 30, 1989.

(12) Williams, L., Pyramidal Parametrics, Computer Graphics, Vol 17, No. 3, July 1983.

(13) Marek, J., Sten, T., "Traffic Environment and the Driver", Charles C. Thomas, Springfield Ill, 1977.

BIOGRAPHICAL NOTE

The author was born 14 April 1942 in Portland, OR. He attended parochial schools and graduated from the University of Portland in May 1964 with a Bachelor of Science degree.

In August 1964, the author began service as a Second Lieutenant in the United States Air Force at the Air Force Institute of Technology, from which he graduated in 1966 with a Master of Science in Space Physics. For the bulk of the next 18 years, he served as an Electrical Engineer in various communications and spacecraft programs. He retired from the Air Force in 1984 as a Major.

The author's nieces and nephews have averaged the destruction of one car each in the process of learning to drive. One was lucky enough to hit the last two feet of guard rail in an uncontrolled skid rather than go over the cliff behind it. This situation indicated a strong need for more realistic driver training. The author completed the requirements for the Master of Science degree in September 1990. He is continuing the simulator project with a view to making it a commercial product.

SIGN EDITOR MANUAL

Synopsis: signed flat

Signed draws the objects described on each line of the ascii file flat.f (if it exists), allows interactive editing of file lines and/or its drawing, and finally stores the revised flat.f and the corresponding binary file flat.F. Up to 127 objects are drawn on a 512 horizontal by less than 496 vertical pixel screen. Binary flats have a fixed size of 256 by 128 pixels, so pixels are reused or combined as needed to adjust the arbitrary input aspect ratio to the fixed flat ratio. Each object entered has independent on and off colors but only 15 arbitrary color pairs and *clear* are allowed in the binary file. Object colors may include a density (probability of coloring each pixel) to give randomized textures to objects.

The objects drawn principally support highway sign construction, and so consist of lines, arcs, text strings and outlines (the rectangular frames with rounded corners around such signs). Arcs and outlines may be filled to produce circles and rectangles with rounded corners. Object descriptions are provided in the first section of Table 1; an example of their use is attached. The descriptions are those used in flat.f files and may also be entered from the keyboard to create new objects. Objects may be modified by entering a replacement object description or object modification command (second table section) while the object to be modified is selected (flashing). Selection is accomplished by depressing the left mouse key with the cursor on the object; current cursor position and the selected object's description are printed when selecting. If the on and off colors are different, flashing due to selection alternates between the colors, otherwise between the given and a contrasting color. Selecting the left mouse key over background removes selection. The middle mouse key is used to place construction marks, which can be used in place of a coordinate pair in braces in the table descriptions. The right mouse key is used to move or copy objects or to copy color; its function is selected by the commands in the last table section. For each function, pressing the right key picks an object/color, and releasing the key places the object/color.

Definitions of Object Description Parameters:

(1) Screen positions and widths are entered manually in displayed pixel units referenced to the upper left corner of the screen with the x (horizontal) coordinate first.

(2) Color is one or more function character/value components as indicated in Table 2, where each value is two decimal characters. The normal color use is to enter a red, green and blue value for a opaque or self-lit surface with an optional density. The density placement pattern is a pseudorandom sequence initialized to the object position.

Table 1. SIGNED Commands

Command	Syntax	Comment
	Object Description Commands	
backgrd	b size_x y bottom_line bck_color	flat's physical size in cm; for auto bottom calc, use 0
arc	a {x y}{radius} quad width color	radius can be mark on arc; quad is sum of 1 for upper right quadrant, 2 for lower left, etc. width of 0 fills the outline
line	l {start_x y}{end_x y} width color	
hunk	h {start_x y}{opposite_x y} color	rectangle aligned with screen
outline	o {right bottom}{left top} w color	default left/top are 0/0; width of 0 fills the outline
text	T {start_x y} color font text	text may include white space
	Object Modification Commands	
text	t text_string	if no object selected, at cursor with last font/color
font	f font	
width	w width	
paint	p color	
	General Commands	
delete	d	delete selected object
undo	u	undo last change
rebuild	r	rebuild if change leaves hole
edit flat	v	call vi on current ascii flat.f
shift all	s x_shift x_attn y_shift y_attn	shift and reduce all objects
quit	q	quit
write	x	write file flat.f & flat.F
make test	y	test flat compression
print test	z	print compression results and quit
	Right Mouse Key Function Selection	
copy object	c	pick object at button push, place copy on release
move object	m	remove object at push, place it on release
group color	g	pick color at button push, change color at release Colors same as picked flash

Table 2. Color Description Components

Component	Syntax	Comments
opaque	oRRGGBB	RR etc are colored reflect coef (0 to 99)
self-lit	sRRGGBB	RR etc are colored intensities (0 to 99)
density	dDD	Set density to DD (0 to 99%)
force	fFF	Force color number to be FF (0 to 15)
clear	c	Color 0 always transparent
tint	tTT[RRGGBB]	Colors 1 to 7 enabled as group for tint (TT: 10 to 70%)
ice	iII[RRGGBB]	Colors 1 to 7 enabled as group for ice (II: 10 to 70%)
glass	g[RRGGBB]	Color 14 enabled as glass (tint of 40%)
mirror	m	Color 15 enabled as mirror

Switched lights are entered as both on and off colors in that order plus the color number to be assigned (forced) for switching purposes. Transparent colors (clear, glass, mirror, ice or tint) may also be entered, which envoke the corresponding forced values and enables. To avoid some potential flat-explosion situations (e.g. mirrors in mirror images), transparents other than clear may be replaced during rendering by opaques simply by removing the enables. Thus these transparents may also be assigned an opaque color - the defaults are light blues.

(3) Font is a string of the form X.YZ where X is the font style letter (Bold, Regular and Italic) and YZ is the numeric size, with the pixel height of a capitalized font letter about twice the numeric size. Only those in the library are available.

Unsightly pixel choices can result from the use of character bitmaps and aspect ratio adjustment. Hence a touchup mode is available during the construction of binary files. Touchup allows pixels in the final flat to be changed after aspect ratio adjustment and before compression. During touchup, the left mouse key picks an existing flat color and the center mouse key places it while down. The right mouse key cycles through several 'brush' shapes as indicated by the cursor shape; the largest square brush is an 'airbrush' that changes pixels randomly in the cursor area. When fixing text, it is useful to make a short line of a color intermediate between foreground and background to use while touching up characters. Touchup mode is ended with a 'q','x','y' or 'z' letter with the assigned general command meaning or an 'r' to return to object manipulation.

Files:

The **C** structure of binary files as attached.

Font files are stored in the directory "/u/ray/ed/font".

Bugs:

Ends of lines are horizontal (line slope>63 °) or vertical (slope<26 °) else 45 ° slopes. Widths are measured along the edge described. The simplest way to cleanup the ends of angled lines is to place background lines over the unwanted parts at the desired angles.

Signed positions should be specified in physical units from the screen center rather than pixels from the top left corner to simplify interaction with *moded.* Objects should be drawn to the flat with its fixed aspect ratio and reflected to the screen with the necessary aspect ratio adjustment, rather than drawing to about the physical aspect ratio and fixing the aspect ratio later, so that adjustment errors are visible immediately.

Characters are expanded vertically by 3 lines to partially fix thin horizontal strokes in the available font bitmaps. Robust outline fonts should be used; they will be necessary if the aspect ratio must be variable as when drawing to flats.

Flat description file example: signal.f

```
b    600   125 129  c
l      0    10   511    10     12  o181818
l      0    10   511    10     12  o242424d50
l     30    33    30     0      3  o303030
l    311    33   311     0      3  o303030
l     31   120    31    30     32  o181818
l    311   120   311    30     32  o181818
a     31    45    10    15     10  f02s930000o450000
a    311    45    10    15     10  f03s930000o450000
a     31   105    10    15     10  f04s009300o004500
a    311   105    10    15     10  f05s009300o004500
a     31    75    10    15     10  f06s939300o454500
a    311    75     9    15      9  f07s939300o454500
l     90    31    90     4      3  o303030
l    250    31   251     4      3  o303030
l     80    47   261    47     34  o006012
T     94    57  f10o727272  B.10  COMPTON
o    262    64    81    30      3  f10o727272
l    491   119   491    60     28  o181818
l    485   125   511   125      4  o181818
l    485    54   511    54      4  o181818
l    491   128   491    49      6  o181818
a   492   105    10    15     10  f08s696969o333333
a   492    75    10    15     10  f09s936000o453000
```

Binary Flat structure description file: flat.h

```
struct FLAT {
      short magic;                          /* magic: 0720 */
      unsigned char s_type;                 /* color switch type */
     enum stypes { self-lit=32,opaque=16,ice=8,mirror=4,glass=2,tint=1 };
      unsigned short color[16][2];      /* color lut entries: 0 on, 1 off */
        /* color bits: 15 self-lit flag, 14-10 red, 9-5 green, 4-0 blue */
      unsigned char ccolor[128*256]; /* max (uncompressed) color array */
        /* file.F files are only as long as needed when compressed */
      };

/** color[0][x] is always clear, so any opaque color values are unused;
      they are used for default flat x and y sizes, in centimeters **/
```

MODEL EDITOR MANUAL

Synopsis: moded [init_file]

Moded builds an image from the flats listed in model files and interactively entered. It uses the flat name and position/orientation/size to render the binary flat.F texture arrays with hidden surface removal plus shadows/tints, one mirror and one ice surface.

Each flat in the model is potentially rendered three times, once in the direct viewing pyramid, once in the mirror-reflected viewing pyramid and once in the ice-reflected nether world. Mirror/ice reflected flats are only rendered when behind a flat with mirror/ice color types in the direct view. The mirror and ice flats should be the first in a scene so that the mirror and ice reflected scenes will not take excessive rendering time. To avoid additional viewing pyramids, mirrors and ice colors are only enabled in the direct view, otherwise they appear as opaque.

Each screen pixel is represented by direct and mirror reflected colors as well as direct and reflected distances for the closest opaque and shadow. Only one shadow intensity is available in the software simulation, and it still requires two shadow arrays and half of the color lut. The model file init_file is called when the program is started to initialize the screen, visual range, lighting and viewpoints. To speed rendering, shadows are not checked on every pixel write, but can be incorporated with the build command. Fog is approximated by modifying pixel colors based on each color's average distance from the viewpoint; thus it requires the screen to be scanned for the average by color. Again for speed, fog is not incorporated after every flat is placed, but on the visual range command. Neither build or visual range is destructive, additional rendering may then be done, and the commands repeated. Build and visual range are updated automatically after a model command, and so are not usually called explicitly. During rendering, jittering of the array sample position is a function of a pixel's range and current visual range; an exactly jittered image when visual range is modified requires rerendering the entire model. Otherwise visual range, ambient light levels and the switch settings for each flat's internal lights may be varied interactively at any time.

Table 1 lists the commands used to render flats. An example of their use in m.init and a model file are appended. Rightmost parameters not entered are defaulted to the values indicated. Parameter definitions are:

(1) X, y and z are integers specifying a flat center's position in centimeters in the world model.

(2) Ax, ay and az are integers each specifying a rotation in degrees about their axis; together they specify an orientation in the world model. Ax and az follow the righthand rule, ay the left. For cylindrically symmetric objects, '*' is used for az and the angle is automatically

calculated to make the flat turn toward the direct viewpoint. To avoid rendering a flat when it is backfacing, preceed its az with '*'.

(3) Sx and sy are floating point flat sizes. A flat's sizes are defaulted to those in the binary flat file's header.

(4) Switches is an octal number where each bit position controls the on/off state of that numbered color in (all) flats. On and off are not required to be self-lit and opaque respectively, opaque and self-lit colors may be attached independently to either state using *signed.* The switch command with no settings toggles all switches.

Bugs:

Because of bitplane limitations, only 16 flats can be rendered at a time. If tints are rendered, only one shadow tint level is rendered (glass and tint>3) and only 8 flats with opaque elements are possible.

Fog affects are based on the average distance(proximity) of a color rather than on each pixel's distance. This is apparent in flats with a large range of distances, such as the street under the eyepoint.

The switch command simultaneously controls switches in all flats just to simplify switch value entry.

MODED Commands

Name	Syntax	Comments
flat	f F x y z az ax ay sx sy	render texture array F.F; positions and angles default to 0
flat with ice	i F x y z az ax ay sx sy	render ice array F.F
direct VP	d x y z az ax ay	render from this position/orientation
reflected VP	r x y z az ax ay	position/orientation relative to direct
model	m file	use lines of the file as commands
build	b	show direct/reflected shadow affects
ambient	a integer	rebuild color lut with ambient[0-99]
switches	s settings(in octal)	rebuild color lut with switches
visual range	v y(in centimeters)	rebuild lut with fog's visual range
clear	c	clear screens, arrays and luts
type	t file	type a file
prg	p prg_init(in octal)	default initialization is random
quit	q	

Initialization file example: m.init

```
c
d  0  0  0      0  0 0
r 12 80 12    165 12 0
s 44
a 84
v 31000
#      inside car
f mirror   8 35 10    -10 -10 5     18 6
```

Model description file example: figure.14

```
#      foreground
i street          0 600 -120    0 90 0    1200 900
f lawn2           0 1374 -77    0 85 0    1200 800
f front2        -73 1743 115    0  0 0    1000 350
f roof2         -73 1860 399    0 45 0    1000 400
#      background
f 58th          325 -900 225   180 0 0     300 106
f signal      -155 -750 210   180 0 0     600 125
f signal.s     -90 -870 180   180 0 0     600 125
#      car parts
f car.shadow 100 900 -115    0 90 0     520 300
f car.r        100  820 -56    0  2 0     396 128
```