LAVA

Lava Advanced Visualization Application

USER'S MANUAL

Sun Microsystems, 1995

CONTENT

current release of LAVA supports all standard 3D visualization functions, NURBS, Texture Mapping, Video Texture Mapping, MPEG-1 video input/output and JPEG video input/output. LAVA can be used for complex 3D model visualization, scene modeling, high-quality rendering, video playback, audio playback, VR stereo display, and video texture mapping.

LAVA supports a special interpretational model development (script) language for animation and walkthrough sequence design. This language is extensible and extremely simple for practical usage. The model language is used to control the GUI programmatically for “hands-off” application control.

The main functionality of the current version of LAVA are as follows:

Input:

- 3D objects supporting *.obj, *.ptc, *.off, *.nu, and *.shed file formats.
- 3D models using the *.mdl file format.
- Video movies in MPEG-1 and JPEG file formats.
- Images in Sun’s rasterfile format.

Output:

- 3D objects in *.obj file format.
- 3D models in *.mdl file format.
- Video movies in JPEG, MPEG-1 file formats.
- 24-bit Stochastic Sampled Images in Sun’s rasterfile format.

Visualization and Modeling Functionality:

- wireframe, solid, markers object representation.
- up to the 32 multiple light sources.
- background modeling.
- depth queuing.;
- transparency.
- motion.
- position.
- raster operations.
- shadows.
- NURBS design and control.
- image texture mapping.
- video movies.
- video texture mapping.
- screen snapshot.

- images and geometry manipulation.
- stereo for VR display environments.
- double buffering.
- performance control/measurement.

2.0 History

LAVA is based on a special-purpose testing framework (Leotool), originally designed to test Sun Microsystems native 3D library XGL and to debug new 3D hardware accelerators. The original testing framework was conceived by Scott R. Nelson of Sun Microsystems and an initial version was implemented by John Lilly, a Stanford University student, who worked as a summer intern for Sun in 1992. Enhancements were made by Scott R. Nelson with some of the NURBS code by Ranjit Oberoi.

The idea of the original testing framework being transformed into a multi-purpose, multimedia application belongs to Yakov Kamen. He was responsible for the new architectural changes. Alex Metzler developed imaging functionality and video input/output subsystems of LAVA. Leon Shirman designed and developed vertex level texture mapping. adaptive texture mapping, and digital video filters. Kirk Brown designed and developed the video and audio controller functionality for MPEG-1 and JPEG support, implemented a new user interface and now is responsible for on-going development shared with Alex Metzler and Johan Hagman.

3.0 LAVA Installation

LAVA is distributed in the two versions: *LAVA User’s Package*, and *LAVA Development Kit*.

To be successfully executed, LAVA requires:

Sparcstation (LX, 5, 10, 20, 100 or later) with minimum 32 Mb RAM and 535 Mb hard drive;

SX, or ZX or latest (TZX, FFB) graphics accelerators;

Sun color monitor;

Solaris2.4, XGL & XIL, SUNWrtvc;

50 Mb swapspace;

Lava User’s Package

In the current release, the Lava User’s Package consists of the following components:

lava release notes;

lava ”zxsetup and ffbsetup” script, which is an example of setting environments;

lava - which is an executable binary;

examples, including several models (mdl) files, public domain 3D objects, 24-bit images and Mpeg-1 movies.

To run LAVA:

cd to /opt/SUNWlava/bin

type “source setupzx” (for ZX and “source setupffb” for Ultra 1 Creator 3D)

type “lava”.

LAVA Development Kit

LAVA development kit consists of necessary information to develop your own application based on LAVA. It consists of LAVA development library, LAVA source code, LAVA include, and examples. In the current release, LAVA is distributed in the form of a compound package called <SUNWlavap>.

To install LAVA Development Kit:

- copy SUNWlavap to user’s directory (for instance /tmp);
- as a ROOT user execute the command:

/usr/sbin/pkgadd -d /tmp SUNWlavap

You will be asked to provide a destination directory or to confirm the default (/opt).

4.0 Starting LAVA

Because LAVA is written in XGL, it will run on Sun compatible workstation with a color display and SX or ZX. or TZX and FFB graphics accelerators. LAVA may be started by typing the program name “lava”. When started with no command line options, it starts up with a 500 by 500 window displaying a colored cube on a black background. Most of LAVA’s functionality will work on any Sun “compatible” workstation.

LAVA has the following command line options:

-single

Start up in single-buffered mode instead of the default double-buffered mode. Use this for 8-bit displays.

-file

Specify the 3D object file to be loaded. This may have any of the following extensions:

.obj for Wavefront Corporation (Silicon Graphics Corporation subsidiary) format;

.off for Digital Equipment Corporation object file format;

.ptc for Parametric Technology Corporation format;

.shad for Sun shader format (ancient);

.mdl for LAVA model script.

Motion

Objects may be moved in three different ways.

4.1 DIALS

On systems with control dials, the object can be moved *using the dials*. The dials are assigned as follows:

dial 0Rotate about the X axis

dial 1Rotate about the Y axis

dial 2Rotate about the Z axis

dial 3Scale the object

dial 4Translate along the X axis

dial 5Translate along the Y axis

dial 6Translate along the Z axis

dial 7Does nothing.

4.2 MOUSE

The object may also be moved *using the mouse*. With the cursor in the graphics window, press the “select” mouse button (usually the left button) and the object can be rotated about the X and Y axis by moving the mouse. Press the “adjust” mouse button (usually the middle button) and the object may be scaled when the mouse is moved up and down. Press the right button and the object can be translated in the screen coordinates. In the current version of LAVA, if the mouse is moving when you let go of the object, it will keep moving in that direction. To stop the motion, either hold the object still for a while or just click once, quickly in the window.

SELECT - rotate about X and Y axis

ADJUST - scale object

MENU - move object along screen coordinates

4.3 MENU CONTROL

The object may also be rotated *using the controls in the motion menu*. These controls allow the object to be rotated either a specified number of frames or continually using the “free run” button. For more information see controls section below.

Raster images and video screen may also be moved by the mouse. In this case, pressing <Shift + Left mouse button> effectively attaches the image to the cursor as it is moved. Releasing the mouse button leaves the image or video at its current position.

5.0 Menu Buttons

There are *five* menu buttons across the top of the frame and one text field (“CUBE”) footer (see

Fig.1).

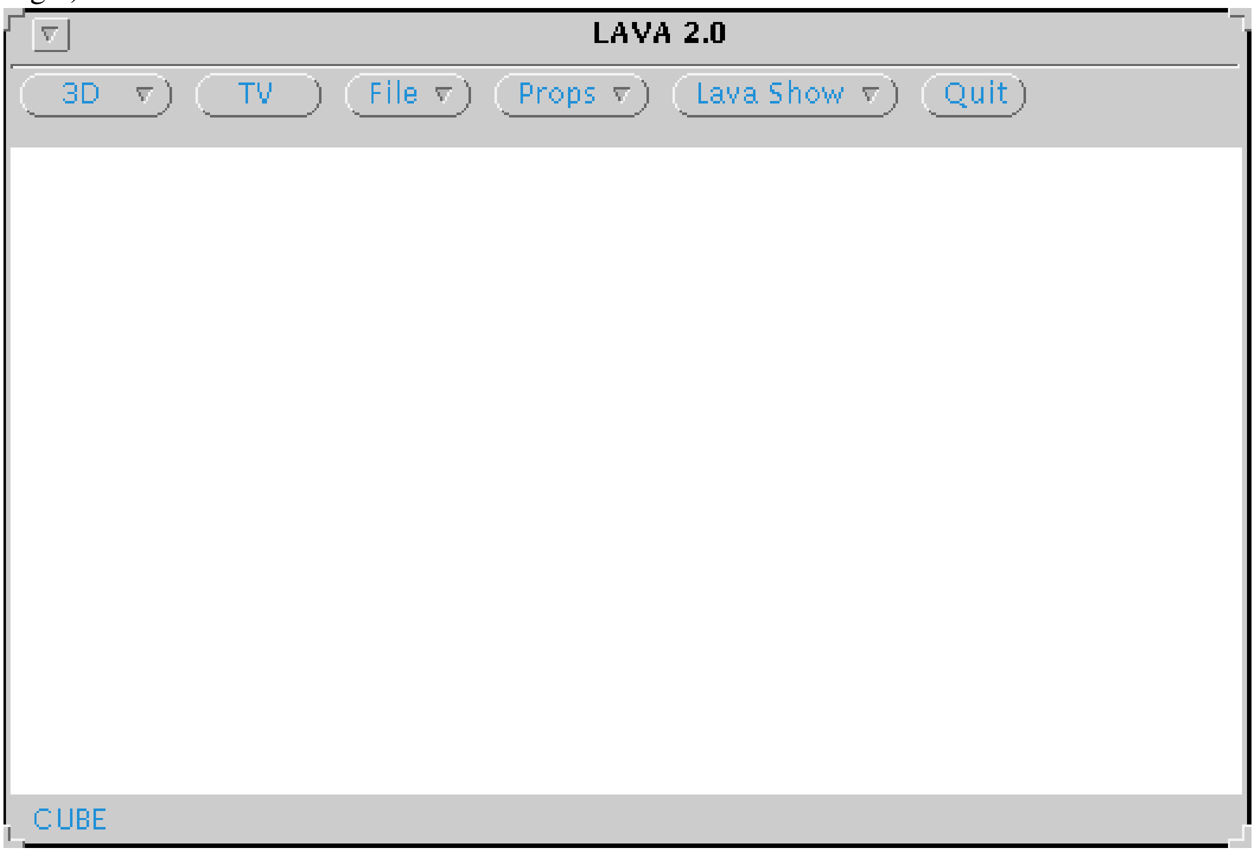


Fig1. LAVA's main window

3D Button:

Designed for 3D visualization control. Pops up a menu with five buttons, three of which brings up submenus. Each menu button brings up a control panel with controls over various attributes and settings. There are more then 20 control panels available to manipulate 3D graphics.

TV Button:

Designed for video and audio media control. Pops up a "VCR-like" control panel.

File Button:

Designed for input/output control. Allows new objects and models to be loaded. Also allows objects, model and image parameters to be saved to disk.

Props Button:

Designed for LAVA support. Contains the performance meter panel, a short help document and an "About" panel listing the authors.

Quit Button:

A simple way to exit the tool.

The text field at the left of the footer displays the current object name and other useful messages.

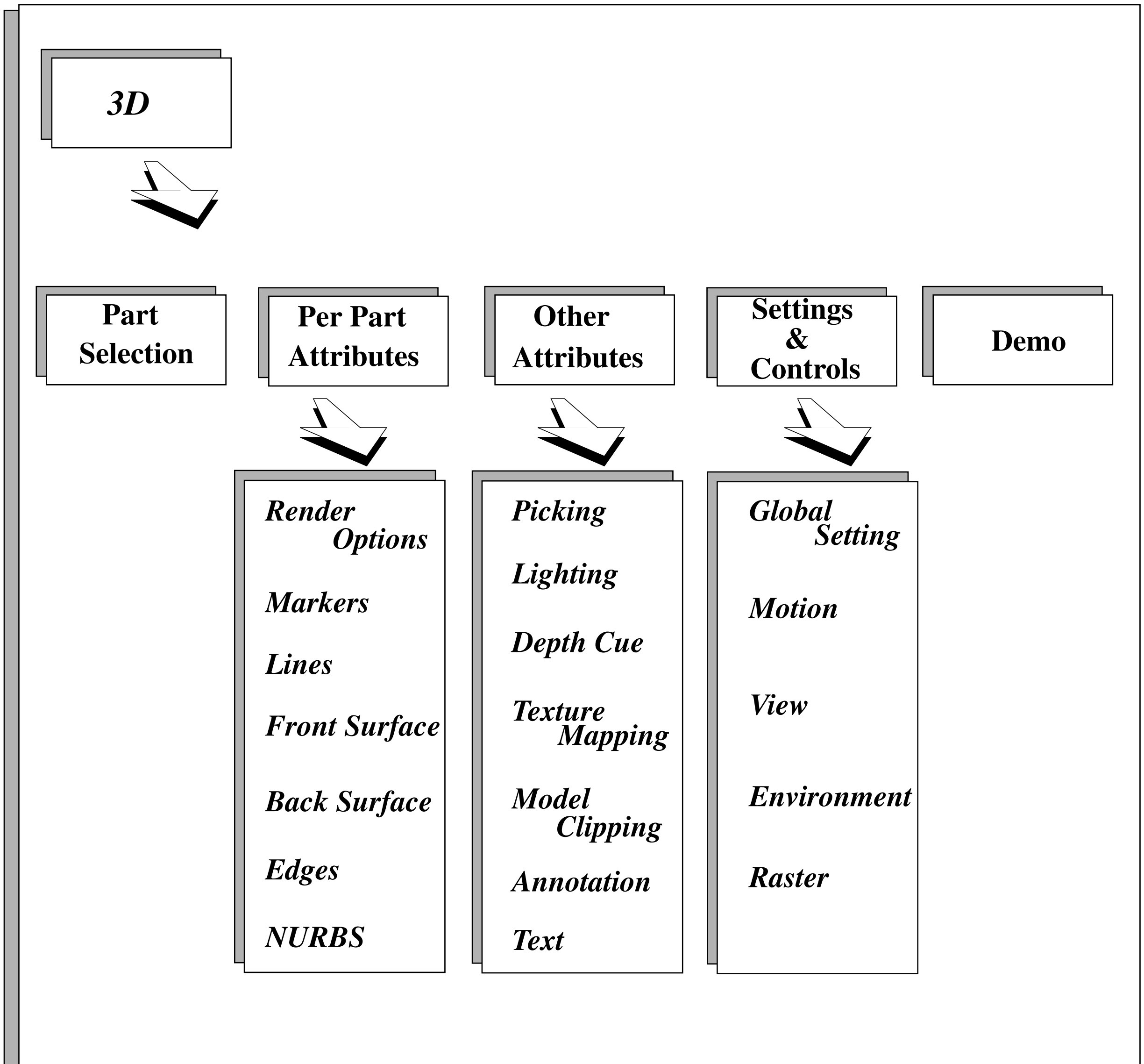


Fig.2 LAVA: 3D Control

6.0 3D Control

The 3D menu allows the user to modify the attributes of the 3D object being displayed.

The 3D menu contains the following five menu buttons (Fig.2):

Part Selection
Per Part Attributes
Other Attributes
Settings & Controls
Demo

6.1 General Functions

Part Selection:

This menu allows user to specify which parts are displayed and which part is the current part. There are the following options in menu: *Select All, Deselect All, Current Part*, and *Object Parts.*

Select All - This button selects all parts to be displayed.

Deselect All- This button deselects all parts so that none of them are displayed. It is assumed the user will then select one part to be displayed.

Current Part- Specifies which is the current part, to which any part-specific attribute settings will apply.

Object Parts- This scrolling list contains the names of all parts in the model. The first item is always "Global", which draws nothing, but affects attributes for all parts. Any attribute for a specific part that has been set by the user or by a model command file will not be affected by global attribute changes. Clicking on a highlighted name turns off the part. Clicking on an un-highlighted name turns on the part and selects it as the current part. To make a displayed part the current part requires a double click. Holding the select button down while moving up and down selects or deselects whichever names are touched.

Note: Select All and *Deselect All* both cause the current part to be "Global".

Per Part Attributes

The "Per Part Attributes" pull right menu contains the following seven menu buttons (Fig.2): *Render Options, Markers, Lines, Front Surface, Back Surface, Edges, NURBS.*

Render Options

This menu is for setting rendering attributes. It may apply for the entire object (Global) or per object part.

Primitive Type-The object or part can be displayed as a solid, wireframe, or as markers at the vertices. Default is Solid.

Face Culling- Face culling off means that both front and back faces will be rendered. Turning culling to Front means that XGL will only draw back faces, and vice versa. Default is Off.

Face Dist(inguishing)- When face distinguishing is off, properties for both front and back surfaces are taken from the Front Surface Properties menu. Turning it on causes the Back Surface Menu to become selectable. Default is Off.

Raster OP- Raster OPallows user to choose which raster operation is performed. RasterOP can be used for special effect design. Default is Copy.

HLHSR Mode- Hidden Line and Hidden Surface Removal can be set on/off. Default is on.

Markers

This menu is used to set marker properties when the object or part is being displayed in

MARKER mode. It may apply for either the entire object (Global) or per part.

Marker Type

Marker Type allows the user to specify which XGL predefined marker is used to represent each point. Choices are: <Dot, Plus, Asterisk, Circle, Cross, Square, Bowtie NE (goes Northeast to Southwest), or Bowtie NW (Northwest to Southeast)>. Default is Dot.

Marker Color

Used to set the marker color, if the Color Selector switch is set to Context. There must be three floats on this line to be recognized. Default is green. (0.0 1.0 0.0)

Antialiasing

For no antialiasing (jaggy lines or dots) turn this off. For markers all by themselves, the best setting is to blend to constant background. For markers and lines mixed with solids, the best setting is to blend to arbitrary background. Default is OFF.

Scale Factor

This determines the size of each marker. It doesn't do anything to the Dot marker. Default is 10.0.

Color Selector

Context means that markers should be the same color as their part color. If vertex colors were implemented, then selecting the Point option would mean that the markers would be the same color as their vertex is specified to be. Default is Context.

Lines

This menu may be used to set line properties when the object or part is being displayed in wireframe mode. It may apply for either the entire object (Global) or per part.

Line Color

Putting exactly three floats on this line allows user to set the line color. Default is green. (0.0 1.0 0.0)

Antialiasing

For no antialiasing (jaggy lines) turn this off. For lines all by themselves, the best setting is to blend to constant background. For lines mixed with solids, the best setting is to blend to arbitrary background. Default is OFF.

Line Width

Allows the user to determine how many pixels wide (or fraction thereof) that lines are drawn. For lines greater than 1.0, the <Endpoint Style> and <Join Style> options become active. Note: Lines wider than one pixel are drawn significantly slower than single pixel lines. Default is 1.0 pixel wide.

<Endpoint Style> - user can select which of three styles are used for endpoints on wide lines: Butt, Square, or Round. Default is Butt.

<Join Style> - user can determine how lines are joined: Device Specific, Beveled, or Mitered. Default is Mitered. When the Join Style setting is set to Mitered, this determines the miter limit. Default is 2.0.

Line Style

<Solid> means that lines will be drawn only in the color specified by the <Line Color> field. Patterned will draw the lines in the color specified by the <Line Color> field and the pattern chosen by the <Line Pattern> selection. <Alt Patterned> will render the lines in a pattern specified by the <Line Pattern> selection and alternating between the colors <Line Color> and <Alt Line Color>. Default is <Solid>.

Line Pattern

Described above in the <Line Style> entry, this may take on the values of {Dotted, Dashed, Dashed-Dotted, Cgm Dotted, Cgm Dashed, Dash-Dot, Dash-Dot-Dotted, or Long Dashed}. Default is Dotted.

Alt Line Color

Field must have exactly three floating-point numbers. The default is white. (1.0 1.0 1.0)

Color Selector

Color Selector specifies whether line colors are determined by their part colors or by their vertex colors. Vertex colors are not currently supported. Default is <Context>.

Color Interpolation Default is OFF.

Front Surface

Front Surface allows user to set surface properties for an object or a part of the object. It may apply for either the entire object (Global) or per part. Back properties only apply when face distinguishing is enabled in the render menu.

Surface color

Specifies the color of the surface. Enter three floats on this line to specify a valid color. Default is white (1.0 1.0 1.0).

Fill Style

Determines how solids are drawn.

<Solid> means that the object is drawn filled with its color.

<Hollow> draws only the edges of each polygon,

<Empty> draws nothing (unless edges are enabled).

<Stipple and Opaque Stipple> render the object with patterns determined by the <Stipple Pattern> setting. Both <Stipple> fill styles will cause a severe degradation in performance, however. Default is< Solid>.

Color Selector

Default is Illumination Independent.

Illum(ination)

Specifies the illumination model:

<None>means no illumination at all,

<Interp Color> interpolates color based on vertex color,

<Per Facet> causes the model to be rendered with flat shading,

<Per Vertex >will turn on Gouraud Shading. Default is Per Facet.

Light Component

Specifies which of the three light components: ambient, diffuse or specular, are used in lighting. Each of the three may be turned on and off independently. The default is all on.

These sliders determine the ambient, diffuse, and specular coefficients, respectively.

Default parameters are: ambient = 0.2, diffuse = 0.8, specular= 1.0.

<Specular Color> specifies the color of the specular highlight. User can entry three float number between 0 and 1 on this line to specify a valid specular highlight color. Default color is white (1.0 1.0 1.0).

<Specular Power> specifies the specular power used in computing the specular highlight. Big number cause small specular highlights and small numbers cause big specular highlights. The specular power changes exponentially between 2.0 and 30.0. Default is 20.0

Transp(arency) Method*:*

Parameter specifies the transparency method used. "None" means no transparency.

It is a default value. "Screen Door" uses screen door transparency. "Blended" uses blended

transparency.

Transp(arency) Blend(ing) Eq(uation):

If the transparency method is "blended", the equation specifies which blend algorithms to be used. "None" does no blending (default). "Constant BG" blends to a constant background. "Arbitrary BG" blends to an arbitrary background (recommended). "Add to BG" adds to the current background.

Transparency:

Transparency slider determines the transparency level. A value of 0.0 is used for fully opaque and a value of 1.0 is used for fully transparent.

Back Surface

Back Surface is identical to Front Surface, but is used for back surface properties.

Transparency method and blend equation are not specified for back surfaces (the front settings are used).

Edges

When the model is rendered as a solid, you may turn edges on to display edges of each polygon and set their attributes in this menu. It may apply for either the entire object (Global) or per object's part.For each object (part) user can *show edge,* specify *edge color*, specify *antialiasing*.

Show Edges

<OFF> means "Do not render polygon edges", <XGL> means "Render edges using XGL API to draw the edges, <Software> means "Render edges using LAVA wireframe to draw the edges". Default is <OFF>.

Edge Color

Parameter specifies the color of the edges. This line must have three floating-point numbers on it (R, G, B) to set the edge color. Default is black (0.0 0.0 0.0).

Antialiasing

For no antialiasing (jaggy lines) turn this <OFF>. For lines all by themselves, the best setting is to< blend to constant background>. For lines mixed with solids, the best setting is to <blend to arbitrary background>. Default is <OFF>.

Edge Width

Parameter allows the user to determine how many pixels (or fraction thereof) in width that edges are drawn. Default is 1.0.

Edge Pattern

Edge Pattern can take the values of {Dotted, Dashed, Dashed-Dotted, Cgm Dotted, Cgm Dashed, Dash-Dot, Dash-Dot-Dotted, or Long Dashed}. Default is <Dotted>.

Alt Edge Color

Alt Edge Color must be three floating-point numbers on this line.
Default is white color (1.0 1.0 1.0).

Edge Style

Parameter's value <Solid> means edges are drawn the color that is specified by the *Edge Color* field. < Patterned> means edges are drawn with the color specified in the *Edge Color* field and the pattern specified by the *Edge Pattern* selection. Parameter's value <Alt Patterned> means =>edges are drawn in the pattern specified by the *Edge Pattern* selection, but alternates colors between *Edge Color* and *Alt.Edge Color.* Note: Both <Patterned> and <Alt Patterned> styles will cause a degradation in performance. Default is < Solid>.

Silhouette. Default is OFF.

SW Offset:

Parameter specifies the offset to use for software edges. The default value is 0.1, which is optimal for most cases.

NURBS

NURBS (Non-Uniform Rational B-Splines) menu is defined for setting multiple NURBSs properties,i.e. *Approx(imation) Method*, *Approx(imation) Val(ue) U, Approx(imation) Val(ue) V, Param(teres) Style, Iso(metrical) Curve Placement, # Iso(metric) Curves U, # Iso(metric) Curves V, Tessellation Method.*

Approx. Method

Parameter specifies whether the NURBS tessellation is done in world coordinates (WC), virtual device coordinates (VDC) or device coordinates (DC), and whether to use metric planar deviation or relative spacing for subdivision. Default is constant parameter subdivision between knots in WC.

Approx Val U

Parameter specifies the approximation value to use in the U direction.

Approx Val V

Parameter specifies the approximation value to use in the V direction.

Param Style

<Iso Curves> shows the curves defining the surface.

<Show Tessellation> draws edges around each generated triangle.

Note: Edges must be enabled in the edge menu before enabling this function.

Iso Curve Placement

Specifies whether isometric curves are placed between knots or between limits.

 # Iso Curves U

Number of isometric curves in the U direction.

 # Iso Curves V

Number of isometric curves in the V direction.

Tessellation Method

Describes methods of NURBS rendering: <Dynamic>, <Combine>, <Static>.
In <Combine> method G-cache is used. In <Static> method no parameters change can be done. <Combine> and <Static> method’s performance is approximately equal.
<Dynamic> method is approximately 4 times slower than <Combine>.
Default is <Dynamic> method

Other Attributes

The Other Attributes pull right menu contains the following seven menu buttons (Fig.2):
Picking, *Lighting*, *Depth Cue*, *Texture Mapping, Model Clipping, Annotation, Text*

Picking

Parameter specifies picking modes and controls. When picking is enabled, the mouse may be used to select a part or facet depending on the picking mode. Use the select button to select a single part. Use the adjust button to get multiple parts. The first part selected (if there are more than one) also becomes the current part (see *Part Selection* to determine “first part”).

When picking is set to *By Facet*, clicking on a single facet will change it to the facet highlight color. This is currently useful with the *Edit* menu.

Picking is available either per part or per facet. Default is <OFF>. Note that cursor motion is not available while picking is enabled.

Pick Aperture

The pick aperture specifies how many pixels in all directions from the center of the cursor qualify to be picked.

Part Highlight Color

Specifies the color used in highlighting the part(s) picked. Highlight color may be set to any desired value.

Facet Highlight Color

Specifies the color used in highlighting the facet(s) picked. If multiple parts or facets fall within the pick aperture all will be selected.

Not e: When picking is disabled, the cursor rotates the object with the select button or changes the size (scales) with the adjust button.

Lighting

This menu is for setting lighting parameters. Up to 32 lights sources may be specified. All lights sources have different default settings, with the first three enabled by default. Light 0 is ambient and lights 1 and 2 are directional. Light 3 is a spot light and light 4 is positional. All the rest are directional with varying colors and directions.

Light #

The number in this field (between 0 and 31, inclusive) corresponds to the light that you are currently working on (and whose attributes are currently displayed). To change the attributes for a specific light, just enter the light number in this field (or use the arrow keys).

Switch:

This determines whether the current light is <ON> or <OFF>. Default is < OFF>.

Type

Determines what sort of light the current light is. Choices are: *Ambient*, *Directiona*l, *Positional*, or *Spot*. Each sort of light has specific attributes which should be set (mentioned in the following text).

Color

Specifies the color of the light by three floats (R, G, B).

Direction

Determines what direction the light is pointing by three floats (R, G, B). Attribute is relevant only to *Directional* and *Spot* lights.

Position

Determines position of the light source by three floats (R, G, B). Valid only for *Positional* and *Spot* lights.

Atten1 & Atten2

Specifies the light attenuation coefficients. Valid only for *Positional* and *Spot* lights. More details are described in {XGL, 95]

Angle

Specifies the spot light angle, in radians, at which, the light is cut off completely. Valid for *Spot* lights only.

Exponent

The spot light concentration exponent. Specifies how fast the intensity falls off as the distance increases Valid only for *Spot* lights.

Depth Cue

This menu sets depth cuing parameters.

Mode

Specifies whether depth cuing is linear, scaled or none. Use scaled for best effect. Default is None.

Color

Determines depth cue color. Default is black (0.0 0.0 0.0) but should be the background color.

Front Ref. Plane

Specifies the position of the front reference plane. The front reference plane is constrained to always be greater than the back reference plane. Default is 0.2.

Back Ref. Plane

Specifies the position of the back reference plane. The back reference plane is constrained to always be less than the front reference plane. Default is -0.2.

Max. Scale

Specifies the ratio of object color to depth-cue color at the front reference plane. Default is 1.0.

Min. Scale

Specifies the ratio of object color to depth-cue color at the back reference plane.

Default is 0.1.

Use BG Color

Determines whether to use the color specified in the second field or to depth-cue to the background color.

Model Clipping

Up to 16 model clipping planes may be specified.

# of Clip Planes

Specifies how many model clip planes are enabled. Default is 0.

Coords

For each model clipping plane a 3D coordinate specifies a point on the plane.

Normals

Specifies a normal to the plane in the direction of the part of the model not to clip away.

Note: the default values for each plane is unique such that adding a plane is visible with most objects.

Annotation

This menu may be used to determine how annotation text is rendered for an object. Annotation names are determined by part names in the data file.

Show Annotation

The object is annotated or not. Default is OFF.

Antialiasing

For no antialiasing (jaggy lines) turn this off. For lines all by themselves, the best setting is to blend to constant background. For lines mixed with solids, the best setting is to blend to arbitrary background. Default is OFF.

Char(acter) Height

Determines how tall the annotation text is. Default is 0.05.

Slant Angle

Determines the slope of the annotation characters. Default is straight up (0.0).

Up Vector

Default is (0.0 1.0).

Color

Determines what color the text will be rendered. Default is green (0.0 1.0 0.0).

Style

The Line option will cause annotation lines to be drawn from the text to the part which it is pointing to. Normal mode will not render those lines. Default is Line.

Pointer Length

This determines how far the text will be from the annotated part. Default is 1.0.

Horizontal Alignment

Specifies how text is lined up horizontally. Choices are: {Normal, Right, Left, and Center}. Default is Normal.

Vertical Alignment

Specifies how text lines up vertically. Choices are: {Normal, Base, Bottom, Top, Half, Cap}. Default is Normal.

Text Path

Determines which direction the text continues from its starting point. Choices are {Right, Left, Up, and Down}. Default is Right.

Settings & Controls

The Settings & Controls pull right menu contains the following 5 menu buttons:

Global Settings
Motion
View
Environment
Raster

Raster menu is used for imaging support and would be discussed in the next chapter.

Global Settings

This menu specifies global settings that don't fit well into other categories. Most may only be set once per frame displayed.

Background Color

Sets the background color for the display window. Enter three floating-point numbers on this line. Default color is black (0.0 0.0 0.0).

Deferral Mode

Determines when things are drawn: As Soon As Possible (ASAP) or At Some TIme (ASTI). ASAP is slower for most accelerators. Default is ASTI.

Double Buffered

Specifies whether drawing will be done in single or double buffered mode. The default setting for this attribute depends on whether or not double-buffering is available on the device.

Stereo Mode

Allows the user to select stereo mode. The object gets drawn twice into the same buffer if stereo is not available. The default depends on the current mode of the window system. If the display is running in stereo mode, this attribute will be enabled.For FFB, exit windows and ffbconfig -res stereo and restart the window system. Now view with stereo headgear.

Reset all attributes

Pressing this button resets all of the global attribute settings. This currently does not update the per-part attributes.

Motion

This menu is used for setting motion/transformation parameters.

Reset All

Resets the local and global matrices to identity. This does not stop motion or reset any motion values. Especially useful when the object has accidentally been translated off the screen and lost.

Stop

Stop all motion.

Start

Begin motion for the specified number of frames. Stop after that many frames.

# of Frames

Specifies how many frames to display after pressing the start button. Most useful for slow hardware or very large models.

Global XYZ-Axis

Specifies the angle of rotation (in degrees) for each axis of the global modeling matrix. Affected by the left (select) mouse button.

Global Scale

Applies a scale to the object, using the global matrix. Affected by the middle (adjust) mouse button.

Global Free Run

Specifies to draw continuously, updating the global matrix each frame.

Global Stop

Stops updates to the global matrix. If the local matrix is not in free run mode, this also stops all automatic motion.

Reset Global Matrix

Resets the global matrix to identity.

Local XYZ-Axis

Specifies the angle of rotation (in degrees) for each axis of the local modeling matrix.

Local Free Run

Specifies to draw continuously, updating the local matrix each frame.

Local Stop

Stops updates to the local matrix. If the global matrix is not in free run mode, this also stops all automatic motion.

Reset Local Matrix

Resets the local matrix to identity.

Note: moving the object using the mouse select button sets the X and Y local axis values and clears the Z axis value. Scaling using the mouse adjust button modifies the local scale setting.

View

This menu specifies how an object is viewed.

Center XYZ

These sliders allow the object center to be moved to get either a more natural appearance or to purposely set the object off center.

Object Center

Specifies how to center the object. To use the slider select Use Slider. To set the center to one of the 6 boundaries select plus or minus XYZ. Minus Y would put a car on its wheels, for example.

Object Scale

Allows a choice of making the object exactly fit within the window (at the narrowest dimension) or to be half that size or to use its original size. Window bounds are -1.0 to 1.0.

Object View Scale

Allows the overall image to be scaled without affecting perspective or other view parameters.

Z-clip Scale

Specifies how the object is scaled in Z to avoid being clipped. This also affects the Z-buffer precision.

Perspective

Adjusts the perspective value.

Environment Settings

Controls additional environment objects.

Floor

To get a floor at the bottom of the object, enable one of these 3 settings. A solid fine grid is best for spot lights. A solid single square is best for other solid floors.

Floor Color

Specifies what color the floor is.

Floor Prim. Type

Specifies <Solid> or <Wireframe>. <Wireframe> floors have less overhead than solid, but don't look as good with shadows.

Floor Position:

Specify the position of the floor relative to the object. A value of 0.0 will touch the bottom of the object. You can slide it up for boats and such.

Shadow

Causes a fake shadow to be drawn. This is accomplished by drawing the entire object again in the shadow color and scaling by 0.0 in Y to squash everything into one plane This is NOT a correct shadow, but looks good with overhead lighting. The shadow is only available when a floor is drawn.

Shadow Darkness

Specifies the percentage of floor color to use in drawing the shadow.

Demo

This special menu has 6 simple controls that are supposed to do the "right thing" without concern for interaction with other controls. All of these will update the appropriate other menus that they affect.

Solid/Wireframe

Switches between solid and wireframe mode.

Line Antialiasing

Turns line antialiasing on and off.

Edges

Turns edges (including NURBS tessellation) on and off.

Transparency

Switches between the two transparency modes and solid rendering.

Depth-cue

Turns depth-cue on and off.

Model Clipping

Turns model clipping on and off.

6.2 Advance Functionality

6.2.1. Imaging

Advance imaging in LAVA is supported by *Raster Settings* menu. Advance imaging allows:

* to import a Sun rasterfile image as a background,
* to snapshot a 3D object and save it as an anti-aliased image,
* to move the background image,
* to undo image saving operations,
* to add a z-coordinate to the image and intersect the image with a 3D object.

This menu controls consists of the following operations:

Left Offset

Slider adds an offset to the left side of the image. It specifies to skip pixels to the left of the image.

Top Offset

This slider adds an offset to the top of the image. It specifies to skip pixels at the top of the image.

Width

Specifies how much of the image to view in the horizontal direction.

Height

Specifies how much of the image to view in the vertical direction.

Switch To Raster

The default mode is *Off*, which means to draw the object is in 3D mode.

Save Pixels

Stores current image as a raster image.

Stochastic

Sample image 8 times with a sub-pixel jittered offset. If the object is in motion mode will create motion blur. Store result image as a raster image.

Reset Image Position

Get the image back in the upper left corner.

Z-buffer

Add z-coordinate to the image, creating 3D image object.

Note: Control+Left mouse button will move image onto the screen.

6.2.2. Texture Mapping

The Texture Mapping controls allow LAVA:

- to map arbitrary sized raster image to any part of the geometrical object;
- to map arbitrary sized image sequences to any part of the geometrical object;
- to choose vertex level or pixel accurate texture mapping method;
- for vertex level texture mapping (VLTM), one of the digital filters can be applied to each part of the object;
- for VLTM subdivision, criteria can be chosen and adaptive subdivision can be implemented.

All necessary information about VLTM is described in [Kamen 95] (see Appendix 1).

The Texture Mapping Control Panel consists of the follows functions:

Texture Source: *OFF, Image, Video*.

LAVA supports two sources of texture maps: images and video. Image texture mapping is a conventional texture mapping, when Sun rasterfile 24-bit images are a source for texture mapping. Different texture maps could be specified for each part of the object.

To view a textured object, a texture map must be previously loaded using the "Load Texture" option of the File menu.

Video can be chosen as a source of the texture mapping. Video source (it can be a movie or video camera) has to be specified from the "TV" Menu. Video can be used for VLTM only.

Binding

In the current release of LAVA automatic binding is performed. Each primitive texture is mapped onto the surface by calculating vertex normals projected onto the surface plane. Additional methods of binding will be specified in the next release of the product.

Texture Method: Vertex-Level, Pixel Level.

Either vertex-level, or standard pixel correct texture mapping can be chosen. Pixel correct texture mapping is supported in software by ZX and TZX acceleration boards and in microcode in FFB board. More information about VLTM is described in Appendix1.

Approx Type

Approximation type is used for vertex-level texture mapping only.

*Num Segment*s specifies into how many segments each triangle side will be subdivided (in model coordinate space).

Pixel Tolerance specifies maximum size of triangle in device coordinate space after adaptive subdivision.

Combined mode selects minimum(Num.Segments; Pixel Tolerance).

Subsegments

The Subsegment's slider is used for vertex-level texture mapping only. It specifies the number of subsegments for Num Segments approximation type.

7.0 TV

7.1 VCR Remote Control

The VCR Remote panel is used to play and save movies to/from the main display window. LAVA will support Mpeg-1 and JPEG movie encoding and decoding. The SunVideo SBus board is used to support a camera device.

The following controls are found on the VCR Remote panel:

Movie - Pops up a menu list from which to select a movie to load, name a file to record a or unload a movie.

Zoom - After the movie is loaded, a small picture is displayed. The window size can be specified.

Camera - Is a toggle button to enable/disable the video stream from the SunVideo board.

Hardware - If the video source is from a camera, laser disk or vcr player various attributes can be controlled here.

- ***Video Board*** selects which board to use.
- ***Video Port*** selects 1 of 3 ports which connects the video device
- ***Video Size*** controls the size of the displayed video window
- ***Quality*** controls the video image quality
- ***Mirror*** changes the video picture perspective to facing left or right
- ***# Frames/Sec*** control video frame drop rate for future networking needs

****Note - Currently multiple video streams are not supported. This will be added in a future release. Also the LAVA expects to see a video signal when "Camera" is activated. So the camera must be ON, and the video player in PLAY/PAUSE.***

SmartPlay - This control is used to provide a "seek" capability for advancing video movie frames.

None - is the default and SmartPlay is inactive

Shuttle - when selected will allow the user to advance or reverse video frames by moving the provided slider back and forth.

Repeat - automatically loops the movie to the first frame until disabled.

Rec - The user must first specify the file to save to by use TV->Movie->Create Movie. Select the number of frames to record by moving the slider. Then click record to save the main display to a movie file. Hit STOP at any time to finish or let LAVA use its default frame count(330) or the number of frames (up to 1000) specified by the slider value.

Play - After the movie file has been loaded, click play to start movie.

Step Reverse - Steps back 1 video frame.

Step Forward - Steps forward 1 video frame.

Rew - Rewinds movie to the first frame.

***Stop* -** Stops the movie.

***FFwd* -** Fast forwards the video in increments of 10 frames. Use the shuttle to jump way ahead.

7.2 Video Texture Mapping

Texture sources can be obtained from either the video camera device attached to the system, or from movies files with the format of *.Mpeg1 and *.Jpeg. When applied with the Texture Mapping attribute “Video”, these video textures are mapped to the specified 3D part. The will provide very realistic scene rendering, depending on the quality of the video source. The resolution and clarity of the image on the 3D object is based on encoding the movie in 24bit depth, recorded video quality and at what size the movie was recorded. For example, if the movie is thumb nail size and the object is zoomed in, the texture map image will be enlarged by increasing the size of the image pixels. Resolution will be lost and the quality of the image will be poor. Another hint, is to record the movie at a faster rate(x3) by using frame drop out rates.

8.0 File Load/Write

The File menu allows the user to load models to be viewed. It also allows models (and attributes), objects and raster images to be saved and reloaded later.

Each of the general load and save control panels, display a scrollable list of potential files or directories to choose from. Double-clicking on the desired file loads that file. Double-clicking on a directory changes to that directory.

The File controls are as follows:

Load Model

Loads in a model script (.mdl). This contains commands to set all attributes as well as to specify the object to be loaded. It may contain motion directives as well. Details of what may go in one of these scripts are found below.

Load Object

Load an object from one of the currently supported object file formats. Wavefront objects are found in files with the .obj extension. DEC Object File Format objects are found in files with the .off extension. Sun Shader files may also be loaded (.shad extension). PTC ProEngineer objects are stored in files with the .slp extension. NURBS files have a .nu or .sof extension. Sun Raster files may also be loaded if they have one of the following extension: .im32, .im24, .im8 or .ras.

Note: that all attributes get reset whenever an new object is loaded.

Load Texture

Loads in a raster file, describing a texture map. If you create custom textures, save to Sun Raster file format in 24 bit depth and save with the file extension (.im24). You can place this in the “Images” directory so it will be available when Load Texture is called.

Save Model

Save the current attribute settings and model specification. This is supposed to save everything so that the exact same image will be seen when reloaded. It is not 100% perfect, but it does work most of the time.

Save Object

Save the current 3D model in Wavefront format. Doesn't work with all models, especially nurbs.

Save Raster

Save the current screen image as a 24-bit Sun Rasterfile. This also switches to raster mode if not already in that mode.

Load Color Cube

Loads a simple cube, identical to that used at when the program is first started.

9.0 Utilities

The properties (utilities) menu currently contains four buttons. Performance data specifies how many of each type of primitives is being displayed, and how fast this data is being displayed.

Edit contains an incomplete edit menu that is still under construction. Help contains a brief overview of how to use LAVA. About contains information about the program authors.

The fields in the Performance Data panel are as follows:

Triangles

The number of triangles being displayed. When this field has a number and the *Lines* and *Points* fields are zero, the primitives per second field is the triangles per second number.

Lines

The number of lines being displayed. When this field has a number and the *Triangles* and *Points* fields are zero, the primitives per second field is the lines per second number.

Points

The number of points being displayed. When this field has a number and the *Triangles* and *Lines* fields are zero, the primitives per second field is the points per second number. If the marker type is other than *Dot*, this number may not have a lot of meaning.

Frames/sec:

Calculate how many times the picture is redrawn per second. The program waits until the time specified by the sample rate has been exceeded, then divides the number of frames by the time that has elapsed. High performance accelerators should be able to draw simple objects at the monitor refresh rate.

Primitives/sec

How many primitives per second are being drawn. Primitives are the sum of the *Triangles*, *Lines*, and *Points*. Two of these three fields should be zero to get a valid performance number for a given primitive type. Note that this number will go up if you switch from double to single buffered mode because the system will no longer wait to synchronize with vertical retrace (but the screen repaint will not appear smooth).

Parts

Specifies how many separate parts the object contains. Each individual part may have separate attributes. An object with more parts has more overhead in the display loop than an object with fewer parts. Otherwise, this number has no direct impact on the primitives per second number.

Facets

How many facets are being displayed. The current version of code (V2.0) turns each facet into a triangle strip with a restart at the beginning. No attempt is made to chain these together into longer strips. This number has no direct impact on the primitives per second number and is only present to provide just a little more information about the model.

Sample rate (sec)

How often the frames per second and primitives per second fields are updated. Because of the overhead of updating the panel, the longer you wait, the more accurate the numbers will be. The default value is 2 seconds, which is a good compromise between the overhead of updating the panel and the attention span of the average viewer. This field may be set by the user.

10.0 Interpretation Script Language

Nearly all attributes that may be set in the Controls menu may also be set in a command script. Command scripts are contained in the model files (.mdl). The command lines must contain a single command per line, although extra white space is allowed anywhere. Commands that require more than one line must have a backslash character (\) as the last character of the line. Comments use the exclamation point character (!) and range to the end of the current line.

A command script may be loaded using the Load Model button on the File menu or the program may be started up under control of a command file as follows:

lava -file test_file.mdl

Note that all but part names are totally case insensitive. Part names MUST match upper and lower case.

The easiest way to get started on creating a .mdl file is to load the desired object, set the attributes you want, then save it using the Save Model command. Then just edit the results until the desired effect is achieved.

In general, any numeric value following an equals sign (=) may also be followed by an operator-equals sign (+=, -=, *=, or /=). This allows loops to be created where values such as color or line widths can change as each image is rendered.

The following list contains all of the currently supported commands, in an order matching the control menu. Any item that allows a choice of several values will have all choices listed in a pair of square brackets (e.g. [off on]). In the command script, exactly one of these choices must be used (and without the brackets). Floating-point and integer values are specified in these examples as <float> and <int> respectively. The numbers in the file may use any format that is acceptable to a standard C compiler. Part names may optionally be encased in double quotes, which are required if the part name has a space character in it.

The following groups of attributes may be set on a global basis or on a per-part basis. To set either the global value or the part value of render.prim_type to solid, any of the following five formats are acceptable:

```
render.prim_type = solid! Sets global primitive type to solid
render.prim_type Global = solid! Sets global primitive type to solid
render.prim_type "Global" = solid! Sets global primitive type to solid
render.prim_type part_1 = solid! Sets part_1 to solid
render.prim_type "part_1" = solid! Sets part_1 to solid
                        ! <= This is a comment, remember?
```

Note once again that part names are case sensitive and may include spaces in some instances.

Attribute commands to turn parts off and on:

part.select <part name>
part.deselect <part name>
part.select_all
part.deselect_all

Attribute setting commands which may optionally be applied per part:

render.prim_type = [wireframe markers solid]
render.face_culling = [off back front]
render.face_dist = [off on]
render.raster_op = [clear set copy copy_inv invert and and_reverse \
    and_inv nand or or_reverse or_inv nor xor equiv noop]
render.hlhsr_mode = [off on]

markers.marker_type = [dot plus asterisk circle cross square \
    bowtie_ne bowtie_nw]
markers.marker_color = <float> <float> <float>
markers.antialiasing = [off constant_bg arbitrary_bg]
markers.scale_factor = <float>
markers.color_selector = [context vertex]

lines.line_color = <float> <float> <float>
lines.antialiasing = [off constant_bg arbitrary_bg]
lines.line_width = <float>
lines.endpoint_style = [butt square round]
lines.join_style = [device bevel miter]
lines.miter_limit = <float>
lines.line_style = [solid patterned alt_patterned]
lines.line_pattern = [dotted dashed dashed_dotted cgm_dotted \
    cgm_dashed dash_dot dash_dot_dotted long_dashed]
lines.alt_line_color = <float> <float> <float>

```
lines.color_selector = [context vertex]
lines.color_interp = [off on]

front_surf.color = <float> <float> <float>
front_surf.fill_style = [solid hollow empty opaque_stipple stipple pattern]
front_surf.stipple_pattern = [horizontal vertical diag_45 diag_135 \
        grid_r grid_d horiz_dbl vert_dbl diag_45_dbl diag_135_dbl \
        grid_r_dbl grid_d_dbl]
front_surf.color_selector = [context facet illum_dep illum_indep]
front_surf.illum = [none interp_color per_facet per_vertex]
! Note: Zero or more of the following three are allowed */
front_surf.light_component = ambient diffuse specular
front_surf.ambient = <float>
front_surf.diffuse = <float>
front_surf.specular = <float>
front_surf.specular_color = <float> <float> <float>
front_surf.specular_power = <float>
front_surf.transp_method = [none screen_door blended]
front_surf.transp_blend_eq = [none arbitrary_bg constant_bg add_to_bg]
front_surf.transparency = <float>

back_surf.color = <float> <float> <float>
back_surf.fill_style = [solid hollow empty opaque_stipple stipple pattern]
back_surf.stipple_pattern = [horizontal vertical diag_45 diag_135 \
        grid_r grid_d horiz_dbl vert_dbl diag_45_dbl diag_135_dbl \
        grid_r_dbl grid_d_dbl]
back_surf.color_selector = [context facet illum_dep illum_indep]
back_surf.illum = [none interp_color per_facet per_vertex]
! Note: Zero or more of the following three are allowed */
back_surf.light_component = ambient diffuse specular
back_surf.ambient = <float>
back_surf.diffuse = <float>
back_surf.specular = <float>
back_surf.specular_color = <float> <float> <float>
back_surf.specular_power = <float>
```

```
edges.show_edges = [off on]
edges.edge_color = <float> <float> <float>
edges.antialiasing = [off constant_bg arbitrary_bg]
edges.edge_width = <float>
edges.edge_style = [solid patterned alt_patterned]
edges.edge_pattern = [ dotted dashed dashed_dotted cgm_dotted \
        cgm_dashed dash_dot dash_dot_dotted long_dashed]
edges.alt_edge_color = <float> <float> <float>
edges.silhouette = [off on]

nurbs.approx_method = [const_param_subdiv_between_knots metric_wc \
        metric_vdc metric_dc chordal_deviation_wc \
        chordal_deviation_vdc chordal_deviation_dc \
        relative_wc relative_vdc relative_dc]
nurbs.approx_val_u = <float>
nurbs.approx_val_v = <float>
! Note: Zero or more of the following three are allowed */
nurbs.param_style = iso_curves show_tess incr_silh_tess
nurbs.iso_curve_placement = [between_knots between_limits]
nurbs.iso_curves_u = <int>
nurbs.iso_curves_v = <int>
```

The following commands only apply on a global basis and may not have a part name keyword applied:

```
picking.picking = [off on]
picking.part_highlight_color = <float> <float> <float>
picking.facet_highlight_color = <float> <float> <float>
picking.pick_aperture = <int>

! Note: To specify light number 4, use “[4]” before the equals sign
lighting.switch [0] = [off on]
```

lighting.type [0] = [ambient directional positional spot]
lighting.color [0] = <float> <float> <float>
lighting.direction [0] = <float> <float> <float>
lighting.position [0] = <float> <float> <float>
lighting.atten_1 [0] = <float>
lighting.atten_2 [0] = <float>
lighting.angle [0] = <float>
lighting.exponent [0] = <float>

depth_cue.mode = [none linear scaled]
depth_cue.color = <float> <float> <float>
depth_cue.front_ref_plane = <float>
depth_cue.back_ref_plane = <float>
depth_cue.max_scale = <float>
depth_cue.min_scale = <float>

texture.switch = [off image video]
texture.binding ! performs texture binding
texture.approx = [num_seg pixel_tol combined]
texture.subseg = <int>
texture.pixel_tol = <int>
texture.tau_func = <int>
texture.update ! updates all texture parameters. Needed before draw command

model_clipping.num_planes = <int>
!Note: To specify clip plane 3, use “[3]” before the equals sign
model_clipping.coord [0] = <float> <float> <float>
model_clipping.normal [0] = <float> <float> <float>

annotation.show = [off on]
annotation.antialiasing = [off constant_bg arbitrary_bg]
annotation.char_height = <float>
annotation.slant_angle = <float>
annotation.up_vector = <float> <float>

```
annotation.color = <float> <float> <float>
annotation.style = [normal line]
annotation.pointer_length = <float>
annotation.horiz_alignment = [normal right left center]
annotation.vert_alignment = [normal base bottom top half cap]
annotation.text_path = [right left up down]

global.background_color = <float> <float> <float>
global.deferral_mode = [asti asap]
global.double_buffered = [off on]
global.stereo_mode = [off on]
global.reset_attr! This is like pressing the button

motion.reset_transforms! These are like pressing the button
motion.stop
motion.start
motion.num_frames = <int>
motion.global_x_rotate = <float>! Note that rotations don't
motion.global_y_rotate = <float>! allow the op-equals format
motion.global_z_rotate = <float>
motion.global_scale = <float>
motion.global_free_run
motion.global_stop
motion.local_x_rotate = <float>! Note that rotations don't
motion.local_y_rotate = <float>! allow the op-equals format
motion.local_z_rotate = <float>
motion.local_free_run
motion.local_stop

view.center_x = <float>
view.center_y = <float>
view.center_z = <float>
view.object_center = [use_slider plus_x minus_x plus_y minus_y plus_z \
        minus_z none]
view.object_scale = [half_screen full_screen original]
view.view_scale = <float>
```

view.z_clip_scale = <float>
view.perspective = <float>

env.floor = [none single_square coarse_grid fine_grid]
env.floor_color = <float> <float> <float>
env.floor_prim_type = [solid wireframe]
env.floor_position = <float>
env.shadow = [off on]
env.shadow_darkness = <float>

Additional motion control commands:

position.reset_transforms

! Reset the local and global position transformation matrices.

position.local_scale = <float>
position.local_x_translate = <float>
position.local_y_translate = <float>
position.local_z_translate = <float>
position.local_x_rotate = <float>
position.local_y_rotate = <float>
position.local_z_rotate = <float>

position.global_scale = <float>
position.global_x_translate = <float>
position.global_y_translate = <float>
position.global_z_translate = <float>
position.global_x_rotate = <float>
position.global_y_rotate = <float>
position.global_z_rotate

A special case exists for matrices:

position.local_matrix = <16 floats>

position.global_matrix = <16 floats>

position.view_matrix = <16 floats>

The matrices are stored when a Model file is saved so that an exact view can be restored. In general, these should be removed from the file with a text editor if you want to create a simple model file or motion script. The view matrix is not used directly, even though it does represent what is used in the program. The various numbers are extracted and matched to the fields from the view menu.

Other commands:

obj_colors.part_color "part name"= <float> <float> <float>

For convenience, this will set the front surface color, line color and marker color for the specified part. This allows colors to be defined that are retained when changing between solid, wireframe and markers.

load_object "object name"

The object name can be a full path or the name of a file in the current directory. The file extension indicates what type of file it is. Doesn't currently work for other .mdl files.

load_texture "texture file"

Loads specified texture file.

draw

Draw the object once. Useful for motion loops.

draw_no_clear

Draw but don't clear the screen. Useful in single-buffered
mode to put multiple objects in a scene.

no_final_move

Sets a flag that prevents a redraw at the end of the .mdl
file when something is drawn using draw_no_clear.

update_menus

For long scripts, this makes sure that any displayed menus
are updated at this point instead of waiting until the
script is fully complete.

loop <int>
end_loop

The loop/end_loop pair loops through the commands in between the number of times specified by the integer value. These may be nested up to 10 levels. It is a good idea to place a "draw" command inside the innermost loop.

quit

Exit from LAVA. Generally used by scripts loaded from
the command line by test programs that would like the program
to exit when the motion is complete.

save_image

Writes the current image out to a file named "ras.im32" in
the current directory. Especially useful with multipass
stochastic sampling (jittering). This could have a better
interface, but was hacked in quickly for V1.2.3.

jitter <int>
end_jitter

Begin a loop using up to 8 internal jitter offsets for multipass stochastic antialising of solid images. This produces a very nice effect, but the current implementation (V1.2.3) is a bit of a hack. The other end of the loop must have an end_jitter command, there must be a draw command somewhere inside the loop, and it is a good idea to enable no_final_move if you want to look at the resultant image. This process is currently extremely slow.

jitter_offset = <float> <float>

Available in case you don't like the predefined jitter values available with the jitter command. Needs a little more work to be useful.

EXAMPLE

The following is a simple motion example showing how a model command file might be built:

```
!
! Make the model move
!
! Best when using the default cube, since it is so simple
!

motion.reset_transforms! Reset all matrices
position.reset_transforms
draw

render.prim_type = solid! Draw a solid object for a while
loop 3
 loop 100        ! Move right while rotating
     position.local_x_translate = 0.002
     position.local_x_rotate = 1.0
     draw
 end_loop
 loop 100        ! Rotate without moving
     position.local_x_rotate = 1.0
     draw
 end_loop
end_loop

position.reset_transforms
render.prim_type = wireframe! Draw wireframe for a while
lines.line_width = 1
loop 5
 loop 40
     position.global_scale = 0.95! Shrink it
     position.global_x_rotate = 2.0
     draw
 end_loop
 lines.line_width += 1.0! Make lines get fatter
 loop 40
     position.global_scale = 1.0526! Expand it
     position.global_x_rotate = 2.0
     draw
 end_loop
 lines.line_width += 1.0
end_loop
```

11.0 Demo Script - assumes beginning state after “lava” is invoked

Show traditional functionality:

rotate cube slightly, stop.
3D->PerPartAttr->RenderOpts->PrimType-> “Wireframe” , rotate cube, stop
3D->PerPartAttr->RenderOpts->PrimType-> “Solid”
File->Load Model->”foot_bones_move.mdl”
File->Load Color Cube

Show video texture mapping:

TV->Movie->Load Movie (choose a movie)
VCR controller panel select SmartPlay “repeat”
Play
3D->Other Attrs->Texture->”Video”
VCR controller Zoom->”1/2x” , Play
Stop, Vcr controller Movie->Unload
Camera (need SunVideo board for this part)
3D->Other Attrs->Texture->”Video”
3D->PerPartAttrs->FrontSurface->TranspMethod->”Blended”
3D->PerPartAttrs->FrontSurface->Transp Blend Eq->”Constant BG”
3D->PerPartAttrs->FrontSurface set Transparency slider to 0.45
3D->PerPartAttrs->FrontSurface->TranspMethod->”None”
Camera (To turn camera off)

Show texture mapping functionality:

rotate cube slightly, stop.
3D->Part Selection->Deselect All
3D->Part Selection->face 3
3D->Part Selection->face 6
File->Load Texture->”Face/madona.im24”
Spin and manipulate single plane. Point out specular lighting.
3D->Part Selection->Select All
Spin Cube and MOUSE DRAG button 3 over the object to the side to create rotation in the Z axis
3D->Settings&Controls->Motion->Reset Global Matrix

12.0 References

[Kamen95] Y. Kamen *Vertex-Level Texture Mapping.* AVS Developers Conference, Boston 1995.
[XGL94] *Solaris XGL 3.1 Programmer's Guide*. SunSoft, 1994.

12.1 LAVA Development Team (present)

Alex Metzler - Designed and Implemented the video input subsystem framework and WWW URL functionality. Currently an active member and contributor. Sun Mircosystems Systems Engineer, Germany.

Kirk Brown - Designed and implemented the VCR controller and digital movie creation functionality, release engineering and packaging. Currently an active member and contributor. Sun Microsystems Engineer; Colorado Springs, USA

Grace Wang - Texture Mapping pipeline contributions and technical support. Sun Microsystems engineer.

Johan Hagman - Active team member for future work. Sun Microsystems Systems Engineer, Sweden.

13.0 Appendix1

Vertex-Level Texture Mapping

Yakov Kamen
Member of Technical Staff, Sun Microsystems Computer Company
2550 Garcia Ave., Mountain View, CA 94043
yakov.kamen@Corp.Sun.COM

Abstract

A new technology of cost-effective texture mapping for Gouraud shading hardware accelerators is proposed. The technology is based on the hierarchical adaptive subdivision of geometrical primitives. Quality/performance trade-off for vertex-level texture mapping implementation is analyzed. Memory management, adaptive subdivision, and control are discussed. The technology is implemented for the Sun Microsystems ZX hardware accelerator.

Introduction

Texture mapping is one of the most interesting and popular technologies today, used to add appropriate realism to artificially designed 3D scenes. Unfortunately, the majority of commercially available texture mapping hardware accelerators are expensive and they are not an element of desktop computers. At the same time, there are several inexpensive accelerated Gouraud shading hardware products. This article describes a solution allowing everyone to use an accelerator supporting Gouraud shading rendering, to accelerate texture mapping.

The general idea of our solution is very simple. For each vertex of the primitive, color is calculated using an appropriate texture mapping algorithm. For each internal point of the primitive, color is calculated using the Gouraud shading algorithm. We called this method vertex-level texture mapping (VLTM), because texture is applied only on the vertices of the primitive, and accurate texture mapping is guaranteed only for the vertices. The interior texturing of the primitive could be inaccurate, but if it is acceptable, nothing else needs to be done. If it is not so, the VLTM method requires a primitive to be subdivided. For each new part (subprimitive) texture is reapplied to the vertices and Gouraud shading is reapplied to internal points. In an extreme case, when only accurate texture mapping is acceptable, the primitive will be finally subdivided into pixel-size subprimitives.

Mathematically speaking, VLTM is a piecewise-linear approximation of a pixel-accurate texture mapping procedure by finite sequence of the Gouraud shading procedures. The quality of this piecewise-linear approximation depends on several parameters: the texture map complexity and size, the primitive size and position in model and device coordinate spaces, and the viewpoint position. The next sections discuss this in detail.

It is difficult to determine who first proposed the combination of Gouraud shading and subdivision as an alternative to the classic accurate texture mapping. A piece-linear approximation of the texture coordinates was used for perspective correction in affine-based texture mapping implementations. Catmull may have already understood this idea in 1974 [Catmull74]. In spite of its simplicity, it was impractical to implement vertex-level texture mapping for almost twenty years. The main drawback of vertex-level texture mapping is the abnormal number of new primitives (usually triangles) generated in the subdivision process when a photorealistic quality is required. Without specially dedicated hardware accelerators, it does not make sense to implement this solution, because the software implementation of VLTM would be slower than classic pixel-accurate texture mapping algorithms. The technology is acceptable only with a graphics accelerator able to perform thousands of Gouraud shading operations per second. The major advantage of VLTM technology is that it fills a gap between a “binary” solution *{Gouraud shading; pixel-accurate texture mapping}* and a multi-value solution *{Gouraud shading; VLTM; VLTM;...VLTM; pixel-accurate texture mapping}.*

Vertex-Level Texture Mapping Pipeline

Vertex-level texture mapping (VLTM) is described by the following algorithm [Shirman 95]:

1. The geometrical object is tessellated by primitives according to a desired criterion.
2. For each vertex of each primitive, surface color and parameter coordinates are obtained.
3. For each vertex, texture coordinates are computed based on obtained parameter coordinates.
4. For each vertex, texture color is computed using current texture coordinates and the current texture map by utilizing a chosen interpolation procedure.
5. For each vertex, texture colors are composed according to a color composition rule [OpenGL92].
6. For each interior pixel of each primitive, a texture color is composed using Gouraud interpolation.
7. The quality of texture mapping is estimated according to considered criteria $\boldsymbol{q}$.
8. If the quality of the texturing does not satisfy the chosen criteria, the surface is retessellated (subdivided) using one of the available subdivision algorithms $\boldsymbol{s}$. For each new vertex in the tessellation, surface color and parameter coordinates are obtained. Return to step 3.
9. If the quality of the texturing satisfies the chosen criteria, the VLTM process is finished.

According to the above pipeline, the image produced by the VLTM method will differ from an image produced by a classic pipeline. The two images would be equal only when quality criteria $\boldsymbol{q}$ requires pixel-accurate texture mapping. They would vary most when quality criteria $\boldsymbol{q}$ requires no additional subdivision and VLTM is a Gouraud shading. There are many interesting intermediate solutions between the two extreme cases, pixel-accurate texture mapping and Gouraud shading, and there are many nontrivial applications for these solutions. Some of them are described below.

Quality/performance trade-off

By sequentially applying subdivision algorithm $\boldsymbol{s}$, we can decrease the size of each primitive, replacing it by several smaller subprimitives. In that case, the quality of the texture mapping increases, but the time to render each primitive decreases. In each concrete case a trade-off can be achieved. This reasoning can be formalized as follows. We will represent each VLTM solution by the triple $< s, q(s), p(s) >$, where s is a subdivision operator, $q(s)$ is a numerical estimation of the texture mapping quality (coefficient), and $p(s)$ is a VLTM performance coefficient. The subdivision operator s is an element of the partially ordered space of the subdivision operator S. A partial order is defined as follows: operator s_0 is less than or equal to operator s_1 if the number of subprimitives $n_p(s_0)$ created by operator s_0 from the original primitive is less than or equal to the number of subprimitives $n_p(s_1)$ created by operator s_1, and the maximum size of the subprimitives created by s_0 is less than or equal to the maximum size of the subprimitives created by s_1. A coefficient $q(s)$ is a measure of texture mapping quality. For simplicity, we can assume that the minimum quality is equal to 0, and the maximum quality is equal to 1. A performance coefficient $p(s)$ is measured from 0 to p_{max}, where p_{max} is maximum performance. The p_{max} depends on each hardware accelerator and can be measured as the number of VLTM textured primitives rendered per second.

In the case of Gouraud shading $s_g = s_{min}$ (no subprimitives have to be generated here), $q(s_g)= 0$, and $p(s_g)= p_{max}$. In the case of pixel-accurate texture mapping $s_a >> s_g$ (subprimitives are pixel size), $q(s_g)= 1$, and $_a)= p_a << p_{max}$. For any other intermediate subdivision s we obtain: $s_g < s < s_a$, $q(s_g) < q(s) < q(s_a)$, and $p(s_a) < p(s) < p(s_g)$. Obviously, $q(s)$ and $p(s)$ are changed in opposite directions, i.e., when $q(s)$ increases, $p(s)$ decreases and visa versa. The concept of quality performance trade-off q/p means a search for an optimal solution for both quality $q(s)$ and performance $p(s)$ (Fig. 1).

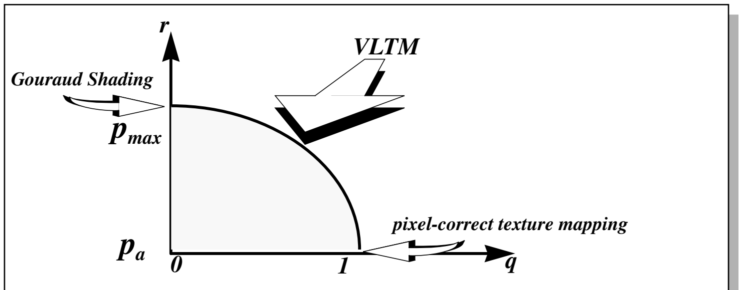


Fig. 1 VLTM quality/performance analysis

How do we compute q/p in practice? As we described above, the quality coefficient q is a complex function of texture mapping complexity i_t, texture mapping size d_t, the primitive's size in model coordinate space x_{MC}, and in device coordinate space x_{DC}, and viewpoint position v. Formally, $q(s) = q(t_t, d_t, x_{MC}, x_{DC}, v)$.

For instance, in the simplest model, $q(s)$ can be considered as a linear function from the primitive's size in model coordinate space x, such that $q(s) = q(i_t, d_t, x, x_{DC}, v) = ax+b$. We assume that for an original primitive x_{MC}: $q(x_{MC}) = 0$ and for each subprimitive x: $q(x_{MC}) < q(x) < q(l_{MC})= 1$, where l_{DC} is the size of the subprimitive on which Gouraud shading and pixel-correct texture mapping are adequate from viewer's point of view. We will call l_{MC} a model tolerance (Color Plate 1). Conventionally, the performance of Gouraud shading graphics accelerator p_{max} is measured in model coordinate space. That is why the performance coefficient $p(s)$ can be simply computed as a function $p(x) = xp_{max} /x_{MC}$. After routine computations we receive an equation for

q/p analysis:

$$p(x) = (1 - rq(x))p_{max}; \qquad r = 1 - l_{MC}/x_{MC} \qquad (1)$$

Using equation (1), for each given quality q or performance p coefficients the necessary size of the subprimitives x or the number of subprimitives $k = x_{MC}/x$ can be calculated. Unfortunately, a subdivision algorithm s for each possible k is unknown today, although several useful heuristic algorithms have been proposed [Kamen95].

In the more complicated model, $q(s)$ can be considered as a linear function from the primitive's size in device coordinate space x, such that $q(s) = q(i_p, d_p, x_{MC}, x, v) = ax+b$. Again, we assume that for an original primitive x_{DC}: $q(x_{DC}) = 0$ and for each subprimitive x: $q(x_{DC}) < q(x) < q(l_{DC}) = 1$. The constant l_{DC} is called pixel tolerance (Color Plate 2). To estimate performance $p(x)$ in device coordinate space is not a trivial task. It can be resolved based on technology created in [Abi-Ezzi91, Shirman 94, 95]. These papers described the function f, which estimates the primitive's size in model coordinate space, based on its size in device coordinate space, and distance from viewpoint. Based on this function we would calculate a performance as: $p(x) = f(x)\, p_{max}/f(x_{DC})$. The q/p equation looks as follows:

$$p(x) = (1 - rq(x))p_{max}; \qquad r = f'(z)(x_{DC} - l_{DC})/f(x_{DC}), \qquad (2)$$

where z belongs to the interval $[x_{DC}, x_{DC}+q(x)(1-x_{DC})]$; as an approximation, we can consider $z = x_{DC}$; f' is a derivative of the function f.

The next level of model complication is generated by adding texture mapping characteristics in the model. Both the model tolerance l_{MC} and the pixel tolerance l_{DC} depend on texture mapping size and complexity. We can consider l_{MC} and l_{DC} adequate to the current texture map's size and complexity if for all $x < l_{MC}$ ($x < l_{DC}$) Gouraud interpolation is equal to the pixel-correct texture mapping [Kamen95].

Memory Management

The VLTM has advantages and disadvantages. One of the big advantages is that it is not necessary to use specially dedicated, very expensive texture mapping memory. Texture maps can be stored in the RAM. A disadvantage is that VLTM uses an additional amount of RAM to generate a set of newly subdivided primitives. In this section we will discuss several methods of memory estimation.

The VLTM memory management depends on the quality $q(s)$ and number of the subprimitives $k(s)$ generated by subdivision s. The total memory M_t used by VLTM can be estimated as follows:

$$M_t = k(s) \text{ x } n_p \text{ x } I + M_{tex}, \qquad (3)$$

where n_p is the number of vertices per primitive (we assume here that all primitives have an equal numbers of vertices), and I is information per each vertex. Usually a vertex can be described in 40–50 bytes (for example: color r,g,b; coordinates x,y,z,w; normals n_x, n_y,n_z; and special data). For triangle primitives $n_p = 3$; for quad primitives $n_p = 4$. Based on equations (1) and (2), memory as a function of q can be computed as follows:

$$M_t = n_p \text{ x } I/(1-rq) + M_{tex}, \qquad (4)$$

where r is the coefficient in (1) and (2).

Equation (4) helps effectively control memory, making restrictions to the quality of texturing:

$$q < (1 - n_p \times I/(M_t - M_{tex})) / r \qquad \textbf{(5)}.$$

Practical Aspects of the Vertex-Level Texture Mapping

The VLTM technology is implemented as an extension to the SunSoft's graphics library [XGL 94] and was tested by Sun's interactive visualization tool called *Leotool.* In Leotool the subdivision algorithm s is implemented in model coordinate space for triangle-based primitives. For a subdivision into m^2 triangles, each edge of the triangle is subdivided by m uniform parts. Three different methods of subdivision control are used. In the first method, subdivision is controlled in the model coordinate space, in the second method, subdivision is controlled in the device coordinate space, and in the third method, subdivision is controlled by combining the two previous methods [Shirman94]. Memory is controlled by equation (5). There are two drawbacks to the current subdivision algorithm: exactly and only m^2 triangles ($m = 1, 2, 3,...$) can be generated by subdivision, and the algorithm works less optimally for triangles with nonuniformly sized edges. Hundreds of images were generated by VLTM in Leotool; several resulting examples are shown in Color Plate 1 and Color Plate 2. For all objects in Color Plate 1, p/q control was applied based on subdivision in the model coordinate space (m is the number of uniform parts). For all objects in Color Plate 2, p/q control was applied based on subdivision in the device coordinate space (l_{DC} is the pixel tolerance).

Conclusion

In this work, we have studied the technology of vertex-level texture mapping (VLTM). Rather than computing texture color per pixel, in VLTM we compute texture color only per vertex of primitives, and simply Gouraud-shade the interior. Vertex-level texture mapping can be used during real-time animations when texturing quality is not very important; better quality can be achieved by finer subdivision. The technique does not required a specially dedicated texture mapping memory, and it is ideally suited to inexpensive graphics accelerators that do not support texture mapping in hardware.

There are several questions which we will address in the future: multi-resolution VLTM technology, video texture mapping, and VLTM usage for digital filtering.

Acknowledgments

This work was sponsored by Sun Microsystems Computer Company. The author would like to thank Dr. Leon Shirman for the invention of device-coordinates-based subdivision control technology and the implementation in Leotool. Yury Kamen's and Rick Goldberg's comments and suggestions were extremely influential.

References

[Abi-Ezzi91] S. Abi-Ezzi and L. Shirman. *The Tessellation of Curved Surfaces under Highly Varying Transformations.* Proceedings of EUROGRAPHICS' 91, 1991, pp. 385–397.

[Catmull74] E. Catmull. *Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D.

thesis, University of Utah, 1974.

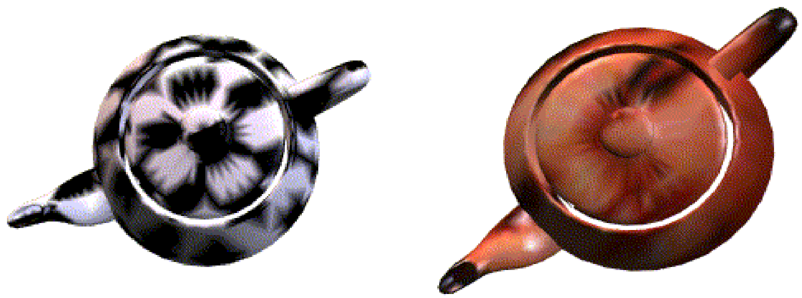
[Kamen95] Y. Kamen and L. Shirman. *Accelerated Cost-Effective Texture Mapping*. Forthcoming.

[OpenGL92] *OpenGL Reference Manual*. Silicon Graphics Inc., 1992.

[Shirman94] L. Shirman and Y. Kamen. *Dynamic Texture Mapping*. Submitted to “Computer Graphics and Applications,” September 1994.

[Shirman95] L. Shirman and Y. Kamen. *Fast and Accurate Texture Placement*. Submitted to SIGGRAPH’ 95, January 1995.

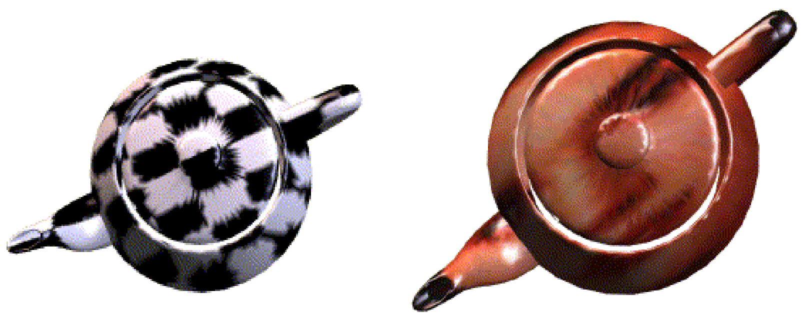
[XGL94] *Solaris XGL 3.1 Programmer’s Guide*. SunSoft, 1994.



1a. Gouraud-shaded teapot:

m = 1;

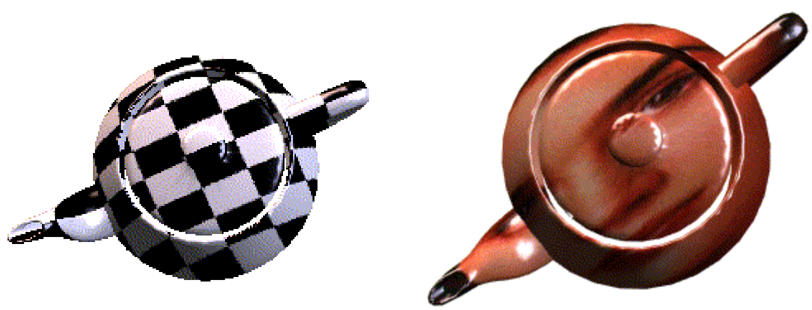
performance: ~90K triangles/s.



1b. VLTM textured teapot;

m = 2;

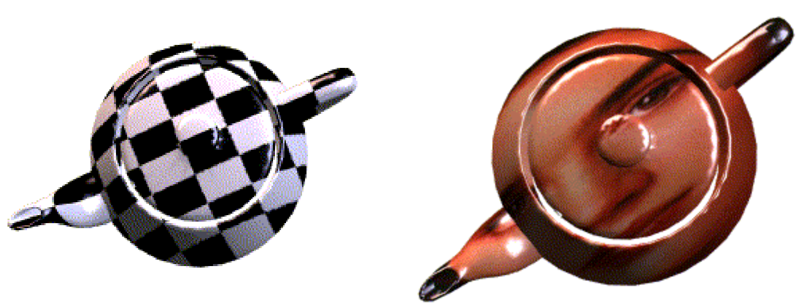
performance: ~30K triangles/s.



1c. VLTM textured teapot;

m = 5;

performance: ~8K triangles/s.

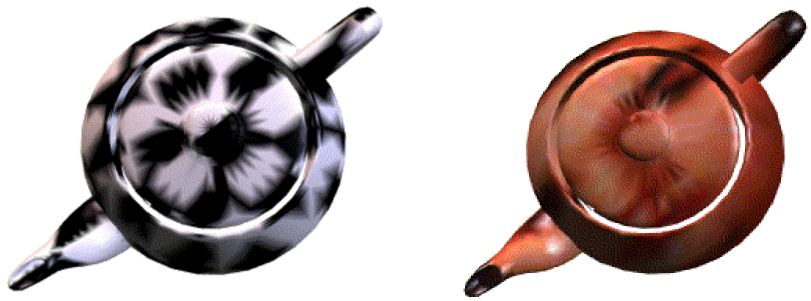


1d. VLTM textured teapot;

m = 9;

performance: ~1.3K triangles/s.

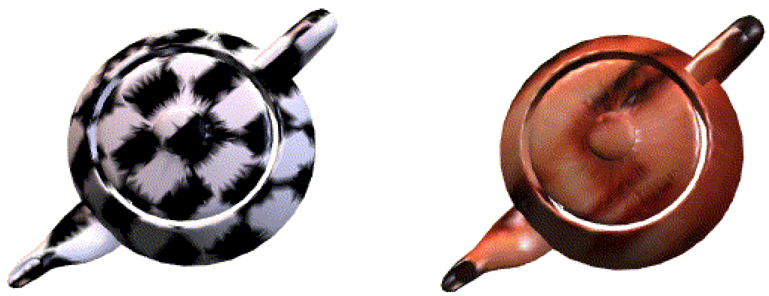
Color Plate 1. VLTM in the Sun Microsystems LAVA on ZX graphics hardware; every triangle is subdivided by m^2 subtriangles in the model coordinate space.



2a. Gouraud-shaded teapot:

$m = 1$;

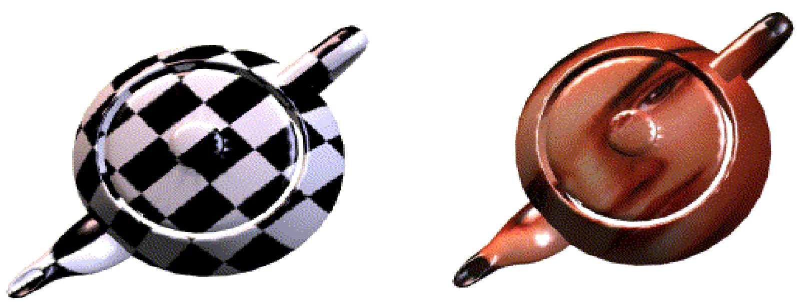
performance: ~90K triangles/s



2b. VLTM textured teapot;

$l_{DC} = 7$;

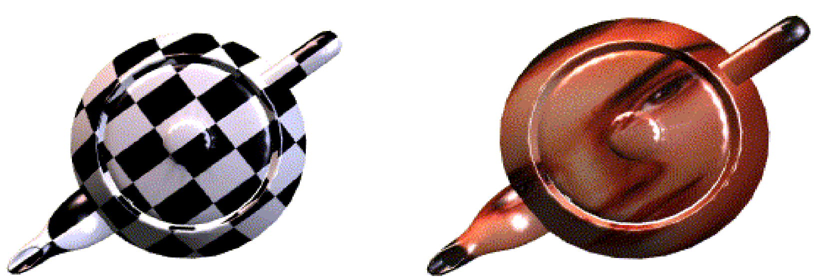
performance: ~ 50K triangles/s



2c. VLTM textured teapot;

$l_{DC} = 3$;

performance: ~11.6K triangles/s



2d. VLTM textured teapot;

$l_{DC} = 2$;

performance: ~ 5.2K triangles/s

Color Plate 2. VLTM in the Sun Microsystems LAVA on ZX graphics hardware;
every triangle is subdivided by l_{DC} pixel tolerance in the device coordinate space.