Video Assistant

Yan Chen 062-86-4795
Jingqian Hu 062-86-9844

Center for Visual Computing
State University of New York at Stony Brook

Abstract

Video Assistant is an OpenGL software that assists in production of high-quality graphics video tapes. This software has a set of 3D polygonal fonts that can be colored, shaded and transformed. A background image, and two light sources are added to improve the visual effect. We also provide different material properties for the string object and different environment ambient lights. Various levels of transparency and fade in or out are realized with cross-dissolve (alpha-blending). In addition, we ported a ripple effect of the background to OpenGL. The only challenge left is that we can not integrate the ripple with the font object because of different projection methods they apply.

1 Using the Program

The program can only run on SGI machines. After untaring vasst.tar, type: "make; make vasst" to produce the executable files. Then type "vasst [string]" to view the wonderful image (String is optional). If the string contains more than one word, please use quota mark to enclose them.

You can click on the buttons of the mouse to view various effects of the title.

- Left Button: you can rotate or translate the string object. By default, it is rotation. You can switch to translation by clicking "translate" on the Motion submenu.
- Middle Button: you can zoom in or out of the string object.
- Right Button: the main menu will pop out. It consisted of 7 submenus and exit control. The 7 submenus include font, material, transparency, effect, environment, light and motion. You can select from any one of them by pressing the right button and draging the mouse.

2 Implementation Details

2.1 Fonts Generation and Loading

Firstly, in vasst.c we check if there is input string. If there is, we use "font3D" to produce the raw triangles of the seven fonts for the string: arial, arialbi, bookos, comic, courbd, times and timesbi. *Font3D* is a utility program for the generation of 3-dimensional text objects. The seven fonts are all defined in *Font3D*. For example, to produce a string of arial font, the command we use is:

font3d font-path = TTF output-path = STRINGS string = "whatever" font = arial.ttf output = string_arial.raw.

In vasst.c, we write these commands into a shell command file: string.com. Then we use "system" call to run it as well as the main display function.

In fact, each font is represented as a specific triangle mesh. We modified "io.c" implemented in Project 2 & 3 to load the triangle mesh into the memory. As we need not convey adjacency information for a triangle mesh efficiently, we remove the redundant attributes for winged-edge data structure. But we also save each vertex and each edge exactly once. And we remove the back surface by using "glCullFace(GL_BACK);".

2.2 Background Image Drawing

In "imgLoad.c", we load the background .ppm image file into the memory.

PPM is short for Portable PixMap format(color), which is a lowest common denominator color image file format. It uses whitespace as part seperator and has the following parts:

- A "magic number" for identifying the file type. A ppm file's magic number is the two characters "P3".
- A width, formatted as ASCII characters.
- A height, again in ASCII decimal.
- The maximum color-component value, again in ASCII decimal.
- Width * height pixels, each three ASCII decimal values between 0 and the specified maximum value, starting at the top-left corner of the pixmap, proceeding in normal English reading order. The three values for each pixel represent red, green, and blue, respectively; a value of 0 means that color is off, and the maximum value means that color is maxxed out.

So the program we wrote to read this format ("imgLoad.c") is as lenient as possible, accepting anything that looks remotely like a pixmap.

As the background image should remain unchanged when the string objects are transformed. In the procedure of drawImgFade(), firstly we use "glPushAttrib(GL_ENABLE_BIT)" and "glPushMatrix()" to save them. Then we set the perspective projection for the image viewing and load the texture image with "gluBuild2DMipmaps()" and "glTexImage2D()". Next, to draw texture-mapped scene, we have to provide both object coordinates and texture coordinates for each vertex[1]. After transformation, the object coordinates determine

[1] OpenGL Programming Guide, Second Edition, pp357-359

where on the screen that particular vertex is rendered. The texture coordinates determine which texel in the texture map is assigned to that vertex. We implemented it with "glVertex()" and "glTexCoord()". Finally, we restore the enable bits and matrices.

2.3 Mulitiple Light Sources

There are two light sources in our image: front light located at {2.0,2.0,2.0,0.0} and back light at {-2.0,2.0,-2.0,0.0}. We can change the color of the front light as selected from the color menu: red, blue, green and white. We also provide the option of turning on off of the front light. The back light is set to green with specular white light to imitate the light reflected from the background of green leaves.

We also change the ambient value of the lighting model to imitate the environment light. There are three options in the environment submenu:

- Default: The RGB values of the ambient light are all set to 0.2 and the alpha value 1.0.
- Up: The RGB values of the ambient light increase gradually from current value to 1.0 by adding 0.1 each frame, while the alpha value remains 1.0.
- Down: The RGB values of the ambient light decrease gradually from current value to 0.1 by reducing 0.1 each frame, while the alpha value remains 1.0.

With each frame, we set with "glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambient_light)", then redraw it.

2.4 Materials and Environment Lights Simulation

Eight kinds of materials are simulated by defining the material properties of the objects of the scene: the ambient, diffuse, specular colors and the shininess. You can select any of the materials from the material submenu: brass, green_gem, grey, drak_yellow, navy_blue, purple, red_plastic and slate. Then we use "glMaterialfv()" to set each component value.

Video Assistant can also display the string with the transparency level specified by user. There are five levels: 0,0 (complete transparent), 0.2, 0.5, 0.8 and 1.0 (complete opaque). We implement it with alpha blending. When user selects a transparency level, the alpha value of the diffuse component of the object material is set to that level's value.

Each time when we display the string object, blending is enabled and the source and destination blending factors are set to GL_SRC_ALPHA and GL_ONE_MINUS_SRC_ALPHA. So we can control how (and how much of) the existing color value should be combined with the new fragment's value. Thus we can use alpha blending to create a translucent string object that lets some of the previously stored image bacdground "show through".

Furthermore, we can change the transparency gradually and continuously to gain the effect of fade in and fade out. We specify the four options in the effect menu:

- Fade to Background: The transparency level of the object decrease gradually from current value to 0 by reducing 0.02 each frame and the background is the green-leaf image.

- Fade from Background: The transparency level of the object increase gradually from 0 to current value by adding 0.02 each frame and the background is the green-leaf image.
- Fade to Black: The transparency level of the object decrease gradually from current value to 0 by reducing 0.02 each frame and the background is black.
- Fade from White: The transparency level of the object increase gradually from 0 to current value by adding 0.02 each frame and the background is white.

With each frame, we set with "glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse[currentMaterial])", then redraw it.

2.5 Title Objects Transformation

Translation is implemented by using "glTranslate()" and rotation by "glRotate()". To zoom in /out, we "translate" the camera. During the process of setting the view point, after "gluPerspective()", we use "glTranslate()" to achieve the effect of translating the camera.

2.6 Ripple Effect Background

We also plan to add the ripple animation effect to the background. The algorithm was designed and implemented by Drew Olbrich in 1992. Now it is included in the software distribution that comes with most SGI machines. It approximates looking down at an image through a layer of water. The user can poke at the water using the mouse, generating one or more ripple patterns. We ported it to OpenGL.

To create the effect, an large texture image is mapped onto a mesh of polygons. Only the texture coordinates of the polygon vertices are distorted – not the vertex coordinates. It includes texture.c, ripple.c and ripple_precalc.c. In texture.c, read_texture() reads in an image file in SGI "libimage" format and converts it to RGBA regardless of the input format and returns the original number of components in the appropriate parameter.

Ripple_precalc.c precomputes the data used to generate the ripple pattern. It generates a very large lookup table for the radial sine wave since that would be too expensive to compute on the fly. A C source file (ripple_table.c) containing the data is produced. This file is compiled and linked into the final executable. In ripple.c, we respond to the mouse pressing (simulation of poking the water) in ripple_click(), look up in the sin wave table in ripple_dynamics() and redraw it in ripple_redraw().

We want to display the ripple background with the string object in the front. But we failed. After careful tracing, we found that: we use perspective projection to draw the string, and parallel perjection to draw the ripple. It seems that we can not have the two kinds of perjection methods at the same time. To make things wore, the string can not be displayed with parallel projection, and the ripple would dispear if perspective projection is assumed.

We tried many ways to integrate them, but none of them work. It may be a furture work for us. As to now, we can only view them separately. There is a switch in main.c: rippleEnable. If you turn it on, and type "make" to compile it, our video assistant will draw the ripple, otherwise if you turn it off, after compilation it showes the string and the background image as we described before.

References

Woo, Mason et al, *OpenGL Programming Guide*, second edition, Addison-Wesley, 1997.

Foley, James D et al, *Computer Graphics: Principles and Practice*, second edition in C, Addison-Wesley, 1996.

Brown, Wayne, C et al, *Graphics File Formats: reference and guide*, Manning Publishing Inc. 1995.

Angel, Edward, *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, Addison-Wesley, 1996.

Appendix: Overview of the Menu

- Font Menu:
  - Arial
  - Arialbi
  - Bookos
  - Comic
  - Courbd
  - Times
  - Timesbi
- Material Menu:
  - Brass
  - Green_Gem
  - Grey
  - Dark_Yellow
  - Navy_Blue
  - Purple
  - Red_Plastic
  - Slate
- Transparency Menu:
  - 0.0
  - 0.2
  - 0.5
  - 0.8

  - 1.0

- Effect Menu:
  - Fade to Background
  - Fade from Background
  - Fade to Black
  - Fade from White

- Environment Light Menu:
  - Up
  - Down
  - Default

- Light Menu:
  - Red
  - Green
  - Blue
  - White
  - Off
  - On

- Motion Menu:
  - Translate
  - Rotate