Lab #2

CSE581 (Spring 2010): Interactive Computer Graphics

Due Date: Sunday, May 2, 2010 by 11:59pm

Guidelines

- Start Early!
- Work on the lab on your own. **No group work**! All suspected cases of academic misconduct will be reported to the University Committee on Academic Misconduct for review.
- Any lab that does not compile will receive a **ZERO**.
- A submission after the 11:59pm deadline is considered late.
- Your lab must compile and run properly on the machines in CL112D.
- If you have written your solution on your home machine or laptop, it is your responsibility to ensure that your solution works as you expect on the machines in CL112D. Allow yourself extra time to port your solution over.
- The grader will NOT overcome compiler or debugging errors for you. Your grade will be based on the compilation and implementation of your solution on the machines in CL112D as is.

Objectives

- Understand the OpenGL 2D rendering pipeline: Set polygon attributes, submit polygons to the graphics pipeline, and place polygons in your scene with affine transformations using the MODELVIEW matrix.
- Input a scene description text file and apply geometric testing to ensure suitable polygons for rendering.
- Animate polygons using the idle callback function.
- Allow panning and zooming over the scene.
- Save snapshots of the scene to a ppm image file.

Description

1. **Read a scene description from a text file into a polygon list data structure.**

2. **Verify that every polygon satisfies the following properties: a minimum of three vertices, a counter-clockwise ordering of the vertices, convex, non-intersecting, and non-collinearity on any sequence of three vertices in polygon.**
3. **Open a rendering window to view your scene. Allow browsing of the scene via panning and zooming of your scene using the mouse.**
4. **Provide a view of your entire scene in a small window at a corner of your rendering window similar to "picture in picture".**
5. **Allow specified polygons to be animated during scene browsing.**
6. **Display the contents of the frame buffer to a ppm image file.**

Assignment

1. ***Read a description of your scene from a text file and then into a polygon list data structure***. Using file I/O commands in C/C++ code, read your scene description from a text file. Create a data structure to store the polygon list with associated attributes for each polygon.

   a. When you run your application, specify the input file on the command line, i.e. **argv** and a**rgc** in your main function.
   b. Assume that the input file will follow the format given in **"lab2InfileFormat.doc"** or **"lab2InfileFormat.pdf"**. If an input file does not conform to this format, then your program may simply exit with or without error.
   c. Use the C functions fopen, fread, and fscanf or C++ file streams to read the file.
   d. Develop your polygon list data structure using data types you are most comfortable for implementation. For example, arrays, structs, the Standard Template Library (STL), etc.
   e. One idea is to store the vertices in an array; store the vertex colors in a second array; store the polygon list in an array of structs, where each struct defines an array to store a sequence of indices into the array of vertices as well as all attributes of the polygon using appropriate data types. If you are using static data structures then you may use the following limits: a maximum of 30 polygons in your scene, a maximum of 100 vertices, and a maximum of 25 vertices per polygon.
   f. Another idea is to define a Class to represent polygons. You can then create polygon objects as needed.

2. ***Valid polygons.*** We will render the scene only if it contains all "valid" polygons, where a "valid" polygon must satisfy <u>all</u> of the following properties:

   a. *At least a triangle*. Check that the polygon has at least three vertices.
   b. *Convexity*
   c. *Counter-clockwise orientation*. Check that the vertices are defined in counter-clockwise order.

d. *Non-intersecting*
e. *No collinearity on three or more successive vertices*

If any polygon in the scene description violates any of these properties then exit *the application*.

To check for conditions ***b*** – ***d*** we will check that every sequence of three vertices of a polygon turns in a counter-clockwise direction using the cross product. We will use a right-handed space and set $z = 0$. To check condition ***e*** we will use the dot product.

The following algorithm checks for conditions ***b*** - ***e***:

```
Given n vertices ($v_0$, …, $v_{n-1}$)
for i = 1 to n – 1
    a = $v_{i-1}$
    b = $v_i$
    if (i < n – 1)
        c = $v_{i+1}$
    else
        c = $v_0$
    vec1 = vector bc, create a 3D vector and set z = 0
    vec2 = vector ba, create a 3D vector and set z = 0
    Normalize vec1 and vec2
    if (vec1 dot vec2 equals -1)
        vertices a-b-c are collinear, so exit program
    vec3 = vec1 cross vec2
    if (z- component of vec3 < 0)
        the polygon is either concave, clockwise, or intersecting, so exit
         program
end for
Polygon passes!
```

3. ***Animating polygons.*** A polygon may be animated in the rendering in any combination of the following three ways:

   a. **Grow/shrink**: A polygon with this attribute turned on will grow and shrink about its center. In its growing phase, have the polygon scale up in increments to a maximum scale factor of '2' and then switch to the shrinking phase where it scales down in decrements to a minimum scale factor of '0.5' and then back to the growing phase and so on. Choose an appropriate increment/decrement value to determine your next scale factor so that the animation is not too fast or too slow.
   b. **Local rotate**: A polygon with this attribute turned on will rotate clockwise or counter-clockwise (depending on the sign of the specified rotation angle) about its center given an angle increment/decrement change you specify.
   c. **Global rotate animation**: A polygon with this attribute turned on will rotate clockwise or counter-clockwise (depending on the sign of the specified

rotation angle) around the world space coordinate axis origin given its angle increment/decrement change you specify.

For grow/shrink and local rotate, you may assume that all polygons in object space are centered about the origin. Polygons that are not may jitter in the animation for certain animation combinations.

A particular polygon may have any combination of these three modes turned on. For example, a polygon may be growing/shrinking, rotating around its center, and rotating around the scene's origin all at the same time.

Apply the appropriate transformations the animation and placing the polygon into world space for each polygon. Determine the order to apply the transformations on a polygon using the MODELVIEW matrix.

Use the idle function callback to drive the animations.

4. ***Scene Viewer.*** Define a rendering window with GLUT to view your scene.

a. Define your rendering window and its functionality as follows:

i. Create a single window with an initial resolution of 512 x 512 using GLUT, give the window a name, and configure the frame buffer to display RGB.

ii. Bind the following keyboard keys to these actions:

1. Pressing 'j' sets the window size to 256 x 256.
2. Pressing 'k' sets the window size to 512 x 512.
3. Pressing 'l' sets the window size to 1024 x 1024

iii. Handle any window resize events.

b. World window default setting, pan, and zoom.

i. We will use OpenGL's default placement of the world window to initialize our world window and use this as our default as well.

ii. Pressing the keyboard key 'n' sets the world window back to the default setting.

iii. Use the left mouse button to **pan**. When the user holds down the left mouse button and drags in any direction, map the horizontal and vertical mouse motions to translations applied to move the world window. For example, when you hold down the left mouse button and drag the mouse to the right, then translate your world window left proportional to the length of the mouse drag. It will appear that you are grabbing the scene and moving it to the right. Perform this for mouse drags in any direction, i.e. horizontal, vertical, and diagonal (this is just horizontal and vertical together).

iv. Use the right mouse button to **zoom**. When the user holds down the right mouse button and drags the mouse up, **zoom in** to your scene proportional to the length of the mouse drag in the vertical direction by scaling the world window size around its center while preserving its aspect ratio. When the mouse is dragged down similarly perform a **zoom out**. Be sure to preserve the aspect ratio.

Determine how to map the length of a mouse drag in pixels to appropriate translations and scales of the world window in world space for ***smooth*** pans and zooms. You may find the following OpenGL commands useful when drawing your polygons: glMatrixMode, glLoadIdentity, glTranslatef, glScalef, and glRotatef.

5. ***Picture in picture (PIP).*** Display a static rendering of the entire scene where the world window is centered at the origin and has a width of 10 units and a height of 10 units. Display this view of the scene at one of the four corners of your rendering window in a second viewport that is much smaller than the GLUT window, say 10% or 20% of the current screen dimensions. This gives the user a global view of the scene as well. First, define your viewport to be the entire screen and draw from your 2D polygon browser's world window. Then, define your viewport for the PIP area and draw from the second world window described above.

   a. Pressing 'm' toggles the display of the PIP viewport to show or not to show this rendering on the GLUT window. At program startup show the PIP.

6. ***Write ppm image file.*** As in Lab #1, save the current frame to a ppm image file called "frame.ppm" when the user presses the 's' key.

7. **Quit.** When the user presses either the 'q' or ESC keys exit the application.

8. **Scene files.** Construct at least <u>three</u> input text files with your own scene descriptions that more interesting than my examples and shows off your implementation.

Image Contest (Extra Credit)

Generate cool images using your solution for the lab for the image contest if you would like to compete for the extra credit.

- Submit your source code and all ppm image files you want us to look at for the image contest.
- If you have separate source code for your extra credit work, then submit this source code in addition to your lab implementation described above.
- Clearly describe in your README text file how to run your code to generate your cool images. Indicate which images you want considered for the contest. ***The grader MUST be able to re-generate your images for the contest from the source code you provide***.

Submission

Follow these instructions carefully to submit the following:

- Use the ***Dropbox*** feature in Carmen to submit your work. You must verify that your submission was successful as follows:
    - You received a confirmation email from Carmen
    - Check the files on Carmen after you submit to make sure the files weren't corrupted in anyway.
- Provide a README text file that contains:
    - Your name
    - Instructions that describe how to compile and run your code (the lab and any extra credit implementation).
    - List all functionality your lab implements correctly.
    - Indicate which images in your submission you want to be considered for the image contest.
- Source code for the assignment
- PPM image files (for the assignment and image contest) that you generated with your source code.
- If you have multiple files to submit, it is preferred that you zip your files into one file and submit just one file.
- **ONLY SUBMIT SOURCE CODE. DO NOT SUBMIT EXECTUABLES!**

Grading

Task	Points
Render scene file into data structure	**20%**
Verify polygons	**15%**
Scene browser: pan, zoom, animate	**50%**
Picture in picture	**5%**
Write ppm file & three scene files	**5%**
Readme file, documented and modularized code	**5%**
Total	**100%**

Suggested Timeline

<u>THESE ARE ONLY SUGGESTIONS AND NOT DUE DATES</u>!

Task	Suggested Date of Completion
Render scene file into data structure	**Wednesday, April 21**
Verify polygons	**Sunday, April 25**
Scene browser: pan, zoom, animate	**Wednesday, April 28**
Picture in picture	