9/10/20

Key-board interaction example in big_deep.cpp
Interaction with the mouse in one the slides

For assignment 1 – define the camera using:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

Note that gluOrtho2D() (omitting the Z coordinates) is not necessarily supported in each version of the vm. Use glOrtho()
e.g., glOrtho(-8, 8, -1, 1, -1, 1) instead of gluOrtho2D(-8, 1, -1, 1)

A Graph

A graph G is defined as $G(V, E, f())$
V - a set vertices ($|V|$ vertices)
E - a set of edges ($|E|$ edges)
$f()$ is a mapping $V \times V \rightarrow$ E ek = (vi,vj)

- A graph is simple (DT) is it has no loops and no double links
- Simple OGL – edges cannot cross
- Convex – if All points on, line segment between two points in a polygon are also in the polygon
- Flat (planner) if all the vertices are on the same plane.

Polygon

- A Polygon is a special case of Graph (Mathematical Graphs / Graph Theory)
- Assume that the edges are “straight” lines
- The edges denote “points” in a coordinate system
- The edges constitute a cycle
- The Polygon is the area enclosed in the cycle.

Valid OGL Polygons
Valid OGL Polygons are simple, convex, and flat
In OGL 3 and above the only valid polygons are triangles.
For complex objects use manual or automatic tiling.

How to do a circle?

For S1 use an n-gon with n >16
Hexa- Penta- octa- gon
n-gon is a polygon with n equal size edges.

Color is a state variable.
Defined using glColor3f(a, b, c);
May need to change prior to glVertex (see slide 5-19)

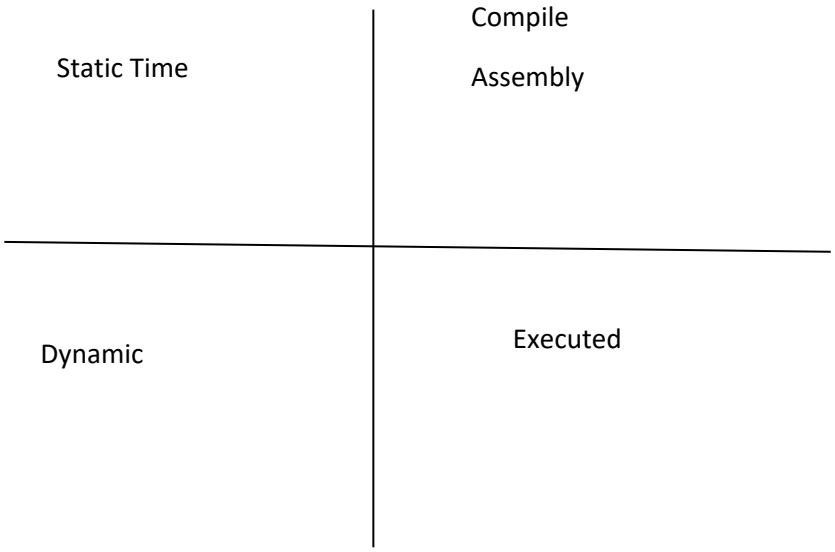
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

GLUT_RGB – use a color buffer.
Color is defined by the gray level of
3 components (R, G, B)
[0,1] - 0 is lowest gray level 1 is highest
[0,255] - 0 lower 255 is the highest

OGL default for color is smooth meaning
interpolation (gradually) of colors between vertices

S0 no need to define a port.
One of the two examples (big_deep) on S0 uses ports.
Use the other example when doing S1.
Ports (5-21)

Static vs. dynamic time



Static time – before the program executes. All the information that is known (to the compiler) before execution.

Most of the GLUT callback functions are executed
Before the program is completely dynamic.

For example setting the window.
It is “almost” static window size location.

The port enables dynamic change of the rendering location
Setting the window is a GLUT function.

Setting the port is an OGL function (can be done within your program)

5-21 assume a window inside the screen (defined using GLUT)
Defined relative to the (0,) of the screen (topmost left most
Monitor pixel)
glutInitWindowSize(500, 500);
glutInitWindowPosition(50, 100);

The port is defined using
glViewport(x, y, w, h)

At any given time. One can define a viewport, where x, y are the location of the port relative to the 0, 0 of the window at (the left most bottom most of the window).
In this case ‘h’ denotes the height of the port and ‘w’ the width.

The frame buffer (FB)
OGL has several memory segments that can serve as frame buffers.

In single mode we use only one of them.
In double mode we use two of them.

Screen (Raster Digital)
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glFlush();

glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glSwap();

The model defined in the model-view (e.g., using glVertex) is constructed (saved) in the frame buffer.

In single mode – glFlush() causes the rendering of the current
contents of the FB to the port/Window/screen

In double mode -Two FBs; one is designated as the front, the other one as the back.
glSwap() renders front onto the screen. Swaps the role of
front and back (front ← → back)

Double why?

1) Avoid artifacts – it takes time to define the objects and render.
   If the image is generated in the FB "slowly"
   then the eye might observe the process.
2) Animation – from frame 2, Build the current frame in back,
   then swap (30 frames per second)
   Using the fact that the eye preserves the screen image for about 1/16
   Of a second.

Consider slide 5-13.

The object is static

The vertices are set at static time (known to the compiler)
glVertex uses constants

How to generate dynamic objects?

Generate objects using variables in glVertex() – for dynamic
How should we feed the values into variables?
GLFLOAT x, y, z ;

- Read from a file,
- Get from the user,
- Get from the KBD,
- Get from the Mouse,

Use a loop or a recursion
To generate the values and place into the variable
Example in sinc.cpp.

Place in a data structure; use the data

glVertex (x, y, z)