Intro to OpenGL Animation Windows and Clipping

CS148: Intro to CG
Instructor: Dan Morris
TA: Sean Walker
June 28, 2005

GL Examples

- Today will be a learn-by-doing sort of lecture… examples are more important than slides
- Example programs will be available on the web
- Today's class will be optimally fun if you encourage me to prod and poke at the code…

Outline for today

- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

OpenGL conventions

- Function names look like:

  gl[action] [#][data type] [v](…)
  - Action tells you what the function does
  - Data type tells you what type (float, double, int, etc.) it works with
  - Number tells you how many it takes
  - A 'v' tells you that this function takes vector (pointer) input
- Top-level documentation only refers to functions by 'action'

OpenGL Data types

Suffix	Data Type	C type	GL type
d	64-bit float	double	GLdouble
f	32-bit float	float	GLfloat
b	8-bit integer	char	GLbyte
i	32-bit integer	int	GLint, GLsizei
ub	8-bit unsigned	unsigned char	GLubyte, GLboolean
s	16-bit int	short	GLshort
us	16-bit unsigned	unsigned short	GLushort
ui	32-bit unsigned	unsigned int	GLuint, GLenum, GLbitfield

Examples

void glVertex3f(GLfloat *x*, GLfloat *y*, GLfloat *z*)

void glColor3b(GLbyte *red*, GLbyte *green*, GLbyte *blue*)

void glMaterialfv(GLenum *face*, GLenum *pname*, const GLfloat **params*)

GL Errors

- Almost all functions return void
- If you want to find out whether there was an error, you need to call glGetError()
- glGetError() is, in technical terms, crazy stupid (editor's opinion)
- It's usually easier to track down your problem without error codes

Outline for today

- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

When do I draw stuff?

- GLUT can give you a callback when the window *needs* to be redrawn
  - glutDisplayFunc, glutReshapeFunc
  - Not useful for animation
- GLUT can give you a callback whenever it's not busy or every few milliseconds
  - glutIdleFunc, glutTimerFunc
- You *can* also draw whenever you want
  - pp1, for example, draws in response to mouse events

An OpenGL Drawing Function

```
void drawMyStuff(void) {

   // Clear the window
   glClear(GL_COLOR_BUFFER_BIT);

   // Do my drawing
   glBegin(SOME_PRIMITIVE_TYPE);
   …
   glEnd();
   glBegin(SOME_OTHER_PRIMITIVE_TYPE);
   …
   glEnd();

   // I'm really done, put my pixels on the screen
   glFlush();
}
```

Drawing GL Primitives

```
// Set up color, texture, location, etc.
glColor3f(1.0f,0.0f,0.0f);

// Tell GL what kind of data to get ready for
glBegin(GL_POINTS);

  // Draw vertices
  glVertex3d(1.0,2.0,5.0);
  glVertex3d(2.0,3.0,10.0);

  // Maybe change some properties
  // and maybe draw some more vertices

// Tell GL you're done drawing for a while
glEnd();
```

Too many slides without a picture

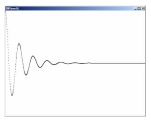
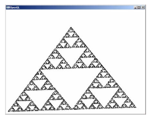
Outline for today

- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

Important GL Primitives: Points

- GL_POINTS
  - Treats each vertex as a single point.
    Vertex n defines point n.
    N points are drawn.
- glPointSize(GLfloat size)
  - Sets the diameter (pixel) of points.
    Subtext: A GL_POINT is not necessarily a pixel

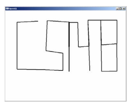
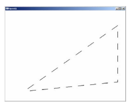
[simple.cpp mathfunc.cpp sier.cpp]

Important GL Primitives: Lines

- GL_LINES
  - Treats each pair of vertices as a line segment. N/2 lines are drawn.
- GL_LINE_STRIP
  - Draws a connected group of line segments from the first vertex to the last. N-1 lines are drawn.
- GL_LINE_LOOP
  - Like GL line strip but it connects the last point to the first point.
- glLineWidth(GLfloat width);
  - Specifies the width of lines (in pixels)

[polyline.cpp stip.cpp]

Important GL Primitives: Polygons

- GL_TRIANGLES
  - Treats each triplet of vertices as a triangle segment. N/3 triangles are drawn.
- GL_TRIANGLE_STRIP
  - Draws a connected group of triangles; each new vertex starting from the third adds a new triangle.
- GL_QUADS, GL_QUAD_STRIP
  - glRect is a shortcut for begin/quad/end
- GL_POLYGON

[polygons.cpp, mystery1.cpp, mystery2.cpp]

Outline for today

- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

Animation 1 (mousemv.cpp)

```
// somewhere in main()
glutMotionfunc(myMouseMove);

void myMouseMove (int x, int y) {

   // put the origin at the bottom-left
   y = screenHeight - mousey;
   glClear(GL_COLOR_BUFFER_BIT);
   make_ngon(x, y, 300, 60);
   glFlush();
}
```

"What's wrong with this approach?"
-Dan Morris, 2005, every single lecture

Single-buffering

- All drawing takes place to a single framebuffer
- Graphics hardware scans buffer whenever it feels like it
- Memory-efficient
- Good for static scenes
- Prone to flickering

Why is flickering worse for large objects?

Double-buffering

- Only update the "real" framebuffer when you're *finished* drawing
- Do all your drawing to a separate framebuffer
- Swap the buffers once per frame
- Terminology:
  - The "front buffer" is shown on the screen
  - The "back buffer" is where you draw

What are some disadvantages of double-buffering?

Double-buffering in OpenGL

GLUT / CS148:

```
// glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

// glFlush();
glutSwapBuffers();
```

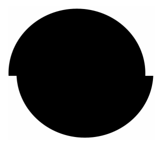
Functions you might see down the road:

```
// For non-GLUT windows GL apps
SwapBuffers(hdc);

// If you need to manually control the current buffer…
glDrawBuffer(GL_BACK); // or GL_FRONT
```

Tearing

- If the buffer-swap happens while you're drawing, can get part of one "finished" frame and part of another
- Solution: hardware makes sure not to buffer-swap while the monitor is refreshing
- Downside: your program blocks



Which way was this circle moving?

Animated OpenGL programs

- glutIdleFunc(): please give me CPU time to think
- glutPostRedisplay(): please call my display function sometime soon
- Recommended software design:
  - Update your virtual world in idle()
  - Do all your drawing in display()

Animation: Using time

- Bad (but common) way to move an object at constant velocity:

```
void idle() {
   object_position += MAGIC_NUMBER;
}
```

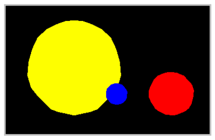
What's wrong with this approach?

```
void idle() {
   double curtime = CS148::getTime();
   double elapsed =  last_time – curtime;
   object_position += MAGIC_NUMBER * elapsed;
   last_time = curtime;
}
```

Animation examples

[animate.cpp rgb.cpp]

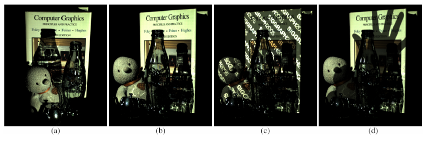
- Also some great hints about what's coming next:
  - 3D positions
  - 3D transformations
  - OpenGL matrix stack



SIGGRAPH video break

Dual Photography, Pradeep Sen at al, SIGGRAPH 2005

- Terms:
  - Helmholtz reciprocity
  - Re-lighting

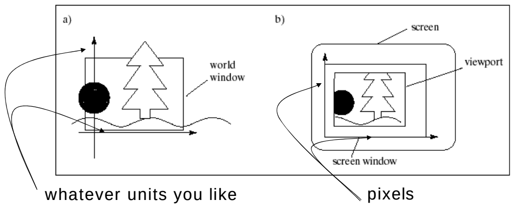


Outline for today

- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

World Windows and Viewports

- *World window*: what part of my model should OpenGL display?
- *Viewport*: where should that image go in the window?

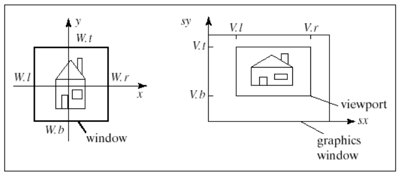


World Windows and Viewports in GL

- gluOrtho2D(left, right, bottom, top)
  - Show the part of my 2D world that lives in this rectangle
- glViewport(left, bottom, w, h)
  - Use only this rectangle within my window
- Games often use multiple viewports

Aspect Ratio

- World window is always mapped to the viewport
- If they don't have the same *aspect ratio* (width / height), the image will be distorted



Outline for today

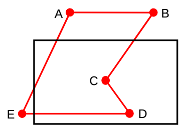
- OpenGL conventions
- OpenGL program structure
- OpenGL primitives
- Animation
- SIGGRAPH video break
- Windows and viewports
- Clipping

Clipping

- Pixels don't get drawn outside the viewport
- A "clipper" takes all primitives that would end up partially outside the viewport and "clips" them so they fit
- Throws them away primitives that are entirely outside the viewport
- Lives between vertex input and the rasterizer
- So why do we care?

Clipping for lines

- A line-clipper:
  - Does nothing for lines in the window (CD)
  - Eliminates lines outside the window (AB)
  - Clips endpoints for lines that are partially in the window (ED, BC, AE)



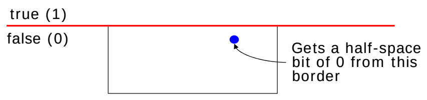
Clipping, take one

- compute line equations (y = mx+b) for the four sides of the clip region;
- for each line to be clipped
  - compute the intersection with each clip region border (two equations, two unknowns)
  - if the line intersects all clip borders outside the box, throw it out
  - ...handle the other cases...

What's wrong with this approach?

Cohen-Sutherland clipping

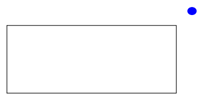
- A border of the clipping region is defined by a line
- This line defines two "half-spaces"
- We'll call the half-space that's outside the clipping region "true"
- A point gets a "half-space bit" from each line



Half-space codes

- The "half-space code" for a point is a 4-bit code containing hs-bits for the four lines
  - We'll put bits in the order (l,r,b,t)

What's the hs code for the blue point?
Where would a point with hs code 0010 be located?



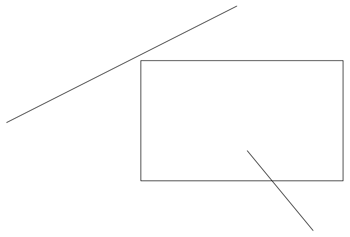
Half-space codes for segments

- Given half-space codes c1 and c2 for two ends of a segment, how do we check the trivial (both endpoints) inside-outside cases?
  - Trivially-inside: (c1 == 0) && (c2 == 0)
  - Trivially-outside: c1 & c2

1001	0001	0101
1000	0000	0100
1010	0010	0010

Lots of possibilities

- Even after we take care of the trivial cases, we could be entirely outside or partially outside



Cohen-Sutherland algorithm

- Clip one point to one edge at a time
- Keep going until you find a trivial case
- Assume we have the clip region stored in variables xmin, xmax, ymin, ymax
- Here's a function to generate a half-space code for a point:

```
int code(float x, float y) {
   return (
     (x<xmin)<<3 | (x>xmax)<<2 |
     (y<ymin)<<1 | (y>ymax)
   );
}
```

```
void clip (float x1, float y1, float x2, float y2) {

    int c1 = code(x1,y1), c2 = code(x2,y2);
    float dx, dy;

    while (c1 | c2)  {
          if (c1 & c2) return;
          dx = x2 - x1;  dy = y2 - y1;
          if (c1) {
                    if (c1 & 8){y1 += dy * (xmin-x1)/dx; x1 = xmin; }
                    else if (c1 & 4){y1 += dy * (xmax-x1)/dx; x1 = xmax; }
                    else if (c1 & 2){x1 += dx * (ymin-y1)/dy; y1 = ymin; }
                    else if (c1 & 1){x1 += dx * (ymax-y1)/dy; y1 = ymax; }
                    c1 = code(x1, y1);
          }  else  {
                    if (c2 & 8){y2 += dy * (xmin-x2)/dx; x2 = xmin; }
                    else if (c2 & 4){y2 += dy * (xmax-x2)/dx; x2 = xmax; }
                    else if (c2 & 2){x2 += dx * (ymin-y2)/dy; y2 = ymin; }
                    else if (c2 & 1){x2 += dx * (ymax-y2)/dy; y2 = ymax; }
                    c2 = code(x2, y2);
          }
    }
    linedraw(x1, y1, x2, y2);
}
```

Next Time

- A bit more OpenGL
  - Display lists
  - Vertex arrays
- 2D Transformations