Previous lesson

- Rasterization
  - line
- circle? => homework
- OpenGL and its rendering pipeline
3 Stages in OpenGL

Define Objects in World Scene

Set Modeling and Viewing Transformations

Render the Scene
Example Code

```c
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(300,300);
    glutCreateWindow("square");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

void display(void)
{
    glClearColor(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex3f(2.0, 4.0, 0.0);
    glVertex3f(8.0, 4.0, 0.0);
    glVertex3f(8.0, 6.0, 0.0);
    glVertex3f(2.0, 6.0, 0.0);
    glEnd();
    glFlush();
}
```
Attribute parameter

• How to generate different display effects?
  • per primitive (C++)
  • system owns states (OpenGL)

• OpenGL is a state machine!
State parameters of OpenGL

- Attributes are assigned by OpenGL state functions:
  - color, matrix mode, buffer positions, Light ...
- on state paras in this lesson
# OpenGL Primitives

- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_QUADS
- GL_POLYGON
- GL_TRIANGLE_STRIP
- GL_TRIANGLES
- GL_QUAD_STRIP

1. GL_POLYGON and GL_TRIANGLE are the only ones in common usage.
2. Valid OpenGL polygons are closed, convex, co-planar and non-intersecting, which is always true for triangles!
Examples

```c
glBegin(GL_POLYGON);
glVertex2i(0,0);
glVertex2i(0,1);
glVertex2i(1,1);
glVertex2i(1,0);
glEnd();
```

```c
glBegin(GL_POINTS);
glVertex2i(0,0);
glVertex2i(0,1);
glVertex2i(1,1);
glVertex2i(1,0);
glEnd();
```
Examples

GLfloat list[6][2] ;

    glBegin(GL_LINES)
    for (int i = 0 ; i < 6 ;i++)
        glVertex2v(list[i]);
    glEnd() ;

    glBegin(GL_LINE_STRIP)
    for (int i = 0 ; i < 6 ;i++)
        glVertex2v(list[i]);
    glEnd() ;

    glBegin(GL_LINE_LOOP)
    for (int i = 0 ; i < 6 ;i++)
        glVertex2v(list[i]);
    glEnd() ;
Examples

GLfloat list[6][2] ;

glColor3f(0.0, 1.0, 0.0);
glBegin(GL_TRIANGLES)
    for (int i = 0 ; i < 6 ;i++)
        glVertex2v(list[i]);
glEnd() ;

glBegin(GL_TRIANGLES)
    glColor3f(1.0, 0.0, 0.0);
    for ( i = 0 ; i < 3 ;i++)
        glVertex2v(list[i]);
glColor3f(1.0, 1.0, 1.0);
    for ( i = 3 ; i < 6 ;i++)
        glVertex2v(list[i]);
glEnd() ;
Examples

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

GL_QUAD_STRIP

Must be planar convex
OpenGL Command Syntax

- All command names begin with `gl`
  
  - Ex.: `glVertex3f( 0.0, 1.0, 1.0 );`

- Constant names are in all uppercase
  
  - Ex.: `GL_COLOR_BUFFER_BIT`

- Data types begin with `GL`
  
  - Ex.: `GLfloat onevertex[ 3 ];`

- Most commands end in two characters that determine the data type of expected arguments
  
  - Ex.: `glVertex3f( ... ) => 3 GLfloat arguments`
glVertex

- All primitives are defined in terms of vertices
  - glVertex2f( x, y );
  - glVertex3f( x, y, z );
  - glVertex4f( x, y, z, w );
  - glVertex3fv( a ); // with a[0], a[1], a[2]
Building Objects From Vertices

- Specify a primitive mode, and enclose a set of vertices in a glBegin / glEnd block

- `glBegin( GL_POLYGON );`
- `glVertex3f( 1.0, 2.0, 0.0 );`
- `glVertex3f( 0.0, 0.0, 0.0 );`
- `glVertex3f( 3.0, 0.0, 0.0 );`
- `glVertex3f( 3.0, 2.0, 0.0 );`
- `glEnd();`
OpenGL Example

```c
void drawOneCubeface(size)
{
    static GLfloat v[8][3];
    v[0][0] = v[3][0] = v[4][0] = v[7][0] = -size/2.0;
    v[1][0] = v[2][0] = v[5][0] = v[6][0] = size/2.0;
    v[0][1] = v[1][1] = v[4][1] = v[5][1] = -size/2.0;

    glBegin(GL_POLYGON);
    glVertex3fv(v[0]);
    glVertex3fv(v[1]);
    glVertex3fv(v[2]);
    glVertex3fv(v[3]);
    glEnd();
}
```
Colors

- OpenGL colors are typically defined as RGB components each of which is a float in the range \([0.0, 1.0]\)

- For the screen’s background:

  - `glClearColor( 0.0, 0.0, 0.0 );` // black color
  - `glClear( GL_COLOR_BUFFER_BIT );`

- For objects:

  - `glColor3f( 1.0, 1.0, 1.0 );` // white color
Other Commands in glBegin / glEnd blocks

- Not every OpenGL command can be located in such a block. Those that can include, among others:
  - glColor
  - glNormal (to define a normal vector)
  - glTexCoord (to define texture coordinates)
  - glMaterial (to set material properties)
Example

```cpp
glBegin( GL_POLYGON );
    glColor3f( 1.0, 1.0, 0.0 ); glVertex3f( 0.0, 0.0, 0.0 );
    glColor3f( 0.0, 1.0, 1.0 ); glVertex3f( 5.0, 0.0, 0.0 );
    glColor3f( 1.0, 0.0, 1.0 ); glVertex3f( 0.0, 5.0, 0.0 );
    glColor3f( 1.0, 0.0, 1.0 ); glVertex3f( 0.0, 5.0, 0.0 );
 glEnd();
```
Polygon Display Modes

-  glPolygonMode( GLenum face, GLenum mode );
-  Faces: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
-  Modes: GL_FILL, GL_LINE, GL_POINT
-  By default, both the front and back face are drawn filled

-  glFrontFace( GLenum mode );
-  Mode is either GL_CCW (default) or GL_CW

-  glCullFace( GlEnum mode );
-  Mode is either GL_FRONT, GL_BACK, GL_FRONT_AND_BACK;
-  You must enable and disable culling with

-  glEnable( GL_CULL_FACE ) or glDisable( GL_CULL_FACE );
Drawing Other Objects

• GLU contains calls to draw cylinders, cones and more complex surfaces called NURBS

• GLUT contains calls to draw spheres and cubes
Compiling OpenGL Programs

• To use GLUT:
  • `#include <GL/glut.h>`
  • This takes care of every other include you need
  • Make sure that glut.lib (or glut32.lib) is in your compiler’s library directory, and that the object module or DLL is also available

• See *OpenGL Game Programming* or online tutorials for details
Structure of GLUT-Assisted Programs

- GLUT relies on user-defined callback functions, which it calls whenever some event occurs
- Function to display the screen
- Function to resize the viewport
- Functions to handle keyboard and mouse events
Event Driven Programming

Main Event Loop

Display Handler

Keyboard Handler

Mouse Handler
Simple GLUT Example

Displaying a square

```c
int main (int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    int windowHandle = glutCreateWindow("Simple GLUT App");
    glutDisplayFunc(redraw);
    glutMainLoop();
    return 0;
}
```
Display Callback

Called when window is redrawn

```c
void redraw()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glColor3f(1, 0, 0);
    glVertex3f(-0.5, 0.5, 0.5);
    glVertex3f(0.5, 0.5, 0.5);
    glVertex3f(0.5, -0.5, 0.5);
    glVertex3f(-0.5, -0.5, 0.5);
    glEnd(); // GL_QUADS

    glutSwapBuffers();
}
```
More GLUT

Additional GLUT functions

```c
glutPositionWindow(int x, int y);
glutReshapeWindow(int w, int h);
```

Additional callback functions

```c
glutReshapeFunction(reshape);
glutMouseFunction(mousebutton);
glutMotionFunction(motion);
glutKeyboardFunction(keyboardCB);
glutSpecialFunction(special);
glutIdleFunction(animate);
```
Reshape Callback

Called when the window is resized

```c
void reshape(int w, int h)
{
    glViewport(0.0,0.0,w,h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0,w,0.0,h, -1.0, 1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```
Mouse Callbacks

Called when the mouse button is pressed

```c
void mousebutton(int button, int state, int x, int y)
{
    if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
    {
        rx = x; ry = winHeight - y;
    }
}
```

Called when the mouse is moved with button down

```c
void motion(int x, int y)
{
    rx = x; ry = winHeight - y;
}
```
Keyboard Callbacks

Called when a button is pressed

```c
void keyboardCB(unsigned char key, int x, int y)
{
    switch(key)
    { case 'a': cout<<"a Pressed"<<endl; break; }
}
```

Called when a special button is pressed

```c
void special(int key, int x, int y)
{
    switch(key)
    { case GLUT_F1_KEY:
        cout<<"F1 Pressed"<<endl; break; }
}
```
OpenGL – GLUT Example

```c
#include <gl/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void init( void )
{
    glClearColor( 0.0, 0.0, 0.0, 0.0 );
    glShadeModel( GL_FLAT );
}

void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glPushMatrix();
    glRotatef( spin, 0.0, 0.0, 1.0 );
    glColor3f( 1.0, 1.0, 1.0 );
    glRectf( -25.0, -25.0, 25.0, 25.0 );
    glPopMatrix();
    glutSwapBuffers();
}
```
OpenGL – GLUT Example

void spinDisplay( void )
{
    spin += 2.0;
    if( spin > 360.0 )
        spin -= 360.0;
    glutPostRedisplay();
}

void reshape( int w, int h )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( -50.0, 50.0, -50.0, 50.0, -1.0, 1.0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
void mouse( int button, int state, int x, int y )
{
    switch( button )
    {
        case GLUT_LEFT_BUTTON:
            if( state == GLUT_DOWN )
                glutIdleFunc( spinDisplay );
            break;
        case GLUT_RIGHT_BUTTON:
            if( state == GLUT_DOWN )
                glutIdleFunc( NULL );
            break;
        default: break;
    }
}
OpenGL – GLUT Example

```c
int main( int argc, char ** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB );
    glutInitWindowSize( 250, 250 );
    glutInitWindowPosition( 100, 100 );
    glutCreateWindow( argv[ 0 ] );

    init();
    glutDisplayFunc( display );
    glutReshapeFunc( reshape );
    glutMouseFunc( mouse );
    glutMainLoop();
    return 0;
}
```
Web Resources

http://www.opengl.org

http://nehe.gamedev.net

http://www.xmission.com/~nate/glut.html
Color and greyscale

• Color is a fundamental primitive attribute

• RGB color model

• Color lookup table / Color map

• Greyscale
Why RGB?
Color Model

[Graph showing light fraction absorbed vs. wavelength (nm)]

[Diagram of human eye with labeled parts: sclera, retina, cornea, crystalline lens, pupil, aqueous humor, iris, ciliary muscles, vitreous humor, blind spot, optic nerve, central fovea]

1.35mm from retina centre

$10,000 \times 10^{-9}$ m

8mm from retina centre

Computer Graphics 2013, ZJU
Color perception

• Three types of cones:
  \[ \begin{array}{ccc}
  S & M & L \\
  \text{Blue} & \text{Green} & \text{Red} \\
  430\text{nm} & 560\text{nm} & 610\text{nm}
  \end{array} \]
  roughly approximate peak sensitivities

• Colorblindness results from a deficiency of one cone type.
OpenGL Color function

- GLUT_RGB and GLUT_RGBA
- alpha channel

- glColor3f (1.0, 1.0, 1.0);
- glColor3i (0, 255, 255);
- glColor3fv (colorArray);
OpenGL Color function

- Color index mode
  - `glIndexi (196);`
- Color blending function
  - `glEnable (GL_BLEND);`
  - `glDisable (GL_BLEND);`
  - `glBlendFunc (sFactor, dFactor);`
OpenGL Color Array

• Defined in the latest OpenGL standard
  • glEnableClientState (GL_COLOR_ARRAY);
  • glColorPointer (...);
• glEnableClientState (GL_VERTEX_ARRAY);
  • glVertexPointer (...);
Attributes of Point and Line

- Point
  - Size and Color
- Line
  - line width
  - line style
  - brush
Region attributes

• defined by a planar polygon

• filling style:
  • wireframe,
  • fill,

• tiling pattern
Polygon filling

- Polygon representation

- By vertex

- Polygon filling:

- vertex representation vs lattice representation

Computer Graphics 2013, ZJU
Polygon filling

• fill a polygonal area $\rightarrow$ test every pixel in the raster to see if it lies inside the polygon.

Question 5: How to Judge…?

Even-odd test

Winding number test
Inside check

\[ wn = \frac{1}{2\pi} \sum_{i=0}^{n-1} \theta_i = \frac{1}{2\pi} \sum_{i=0}^{n-1} \arccos \left( \frac{PV_i \cdot PV_{i+1}}{|PV_i| |PV_{i+1}|} \right) \]

Question 6: How to improve ...?
Inside check

even-odd test
Scan Line Methods

- Makes use of the *coherence* properties
  - Spatial coherence: Except at the boundary edges, adjacent pixels are likely to have the same characteristics
  - Scan line coherence: Pixels in the adjacent scan lines are likely to have the same characteristics
- Uses intersections between area boundaries and scan lines to identify pixels that are inside the area
Scan Line Method

• Proceeding from left to right the intersections are paired and intervening pixels are set to the specified intensity

• Algorithm

  • Find the intersections of the scan line with all the edges in the polygon

  • Sort the intersections by increasing X-coordinates

  • Fill the pixels between pair of intersections

Discussion 5: How to speed up, or how to avoid calculating intersection
Efficiency Issues in Scan Line Method

- Intersections could be found using edge coherence
  the X-intersection value $x_{i+1}$ of the lower scan line can be computed from the X-intersection value $x_i$ of the preceding scanline as

  $$x_{i+1} = x_i + \frac{1}{m}$$

- List of active edges could be maintained to increase efficiency
- Efficiency could be further improved if polygons are convex, much better if they are only triangles
Special cases for Scan Line Method

- Overall topology should be considered for intersection at the vertices.
- Intersections like $I_1$ and $I_2$ should be considered as two intersections.
- Intersections like $I_3$ should be considered as one intersection.
- Horizontal edges like $E$ need not be considered.
Advantages of Scan Line method

• The algorithm is efficient
• Each pixel is visited only once
• Shading algorithms could be easily integrated with this method to obtain shaded area
Seed Fill Algorithms

- Assumes that at least one pixel interior to the polygon is known
- It is a recursive algorithm
- Useful in interactive paint packages
Aliasing

- Aliasing is caused due to the discrete nature of the display device.
- Rasterizing primitives is like sampling a continuous signal by a finite set of values (point sampling).
- Information is lost if the rate of sampling is not sufficient. This sampling error is called aliasing.
- Effects of aliasing are
  - Jagged edges
  - Incorrectly rendered fine details
  - Small objects might miss
Aliasing(examples)

Original

Rendered

Jagged profiles
Aliasing (examples)

Disintegrating textures
Aliasing (examples)

Original  Rendered

Loss of detail
Antialiasing

- Application of techniques to reduce/eliminate aliasing artifacts
- Some of the methods are
  - increasing sampling rate by increasing the resolution. Display memory requirements increases four times if the resolution is doubled
  - averaging methods (post processing). Intensity of a pixel is set as the weighted average of its own intensity and the intensity of the surrounding pixels
  - Area sampling, more popular
Antialiasing (postfiltering)

How should one supersample?

Taking 9 samples per pixel
Area Sampling

- A scan converted primitive occupies finite area on the screen.
- Intensity of the boundary pixels is adjusted depending on the percent of the pixel area covered by the primitive. This is called weighted area sampling.
Area Sampling

- Methods to estimate percent of pixel covered by the primitive
  - subdivide pixel into sub-pixels and determine how many sub-pixels are inside the boundary
  - Incremental line algorithm can be extended, with area calculated as

\[
\text{Area} = m \times x - y + c + 0.5
\]
预祝大家节日快乐！