Objectives

- OpenGL drawing primitives and attributes
- OpenGL polygon face and culling
- OpenGL hidden-surface removal
- Constructing 3D models in OpenGL
- Collision detection
Review: OPENGL DRAWING PRIMITIVES

```c
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();
```

• OpenGL polygons must be convex
void glBegin(GLenum mode);

mode:

• GL_POINTS individual points
• GL_LINES pairs of vertices interpreted as individual line segments
• GL_POLYGON boundary of a simple, convex polygon
• GL_TRIANGLES triples of vertices interpreted as triangles
• GL_QUADS quadruples of vertices interpreted as four-sided polygons
• GL_LINE_STRIP series of connected line segments
• GL_LINE_LOOP same as above, with a segment added between last and first vertices
• GL_TRIANGLE_STRIP linked strip of triangles
• GL_TRIANGLE_FAN linked fan of triangles
• GL_QUAD_STRIP linked strip of quadrilaterals
Review: Points and Line Attributes

- void glPointSize(GLfloat size);
- void glLineWidth(GLfloat width);
- glLineStipple(1, 0x3F07);
  glEnable(GL_LINE_STIPPLE);
- void glLineStipple(GLint factor, Glushort pattern);
Review: SPECIFYING A COLOR

glColor3f(0.0, 0.0, 0.0);
draw_object(A);
draw_object(B);
glColor3f(1.0, 0.0, 0.0);
draw_object(C);

glColor3f(0.0, 0.0, 0.0);  black
  glColor3f(1.0, 0.0, 0.0);  red
  glColor3f(0.0, 1.0, 0.0);  green
  glColor3f(1.0, 1.0, 0.0);  yellow
  glColor3f(0.0, 0.0, 1.0);  blue
  glColor3f(1.0, 0.0, 1.0);  magenta
  glColor3f(0.0, 1.0, 1.0);  cyan
  glColor3f(1.0, 1.0, 1.0);  white
**DEPTH BUFFER (Z-BUFFER)**

```c
glClearColor(0.0, 0.0, 0.0, 0.0); // RGBA, A=>alpha

glEnable(GL_DEPTH_TEST);
...

while (1) {
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    draw_3d_object_A();
    draw_3d_object_B();
}
```

Color buffer

Depth buffer
Clearing the buffers

```c
int depthBuffer(XMAX, YMAX);
Color frameBuffer(XMAX, YMAX);

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) {
  for (y=0; y<YMAX; y++)
    for (x=0; x<XMAX; x++) {
      draw_frameBuffer(x,y);
      // glClearColor
      draw_depthBuffer (x,y, back);
      // Back_Clipping_Plane
    }
}
```
THE Z-BUFFER ALGORITHM

float pixelDepth;
int depthBuffer(XMAX, YMAX);
Color frameBuffer(XMAX, YMAX);

for (each polygon)
for (each pixel (x,y) in the polygon scan-conversion) {
    if (GL_DEPTH_TEST enabled) zBuffer(x,y);
    // glEnable(GL_DEPTH_TEST);
    else draw_frameBuffer(x,y);
}

zBuffer(int x, int y) {
    pixelDepth = polygon’s z-value at pixel (x,y);
    if (pixelDepth < read_depthBuffer(x,y)) {
        draw_depthBuffer(x, y, pixelDepth);
        draw_frameBuffer(x,y); // glColor
    }
}
Polygon’s z-value

• Modify polygon-scan conversion algorithm: for each pixel (x, y)
  – \( Ax + By + Cz + D = 0 \)

• \( z = f(x, y) = - (Ax + By + D)/C \)
  – \( zi+1 = zi + \delta \) //for a scan-line

\( \square \delta = \frac{-A}{C} \)
Depth Test: J2_5_Cone

// clear both framebuffer and zbuffer
    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

// for testing hidden-surface removal
if (cnt % 600 < 500) {
    gl.glEnable(GL.GL_DEPTH_TEST);
}
else  gl.glDisable(GL.GL_DEPTH_TEST);
Faces of a Polygon and Back-face Culling

• Front face polygon has counterclockwise vertices, with normal facing viewer. In OpenGL, this can be defined.

• void glPolygonMode(GLenum face, Glenum mode);

  face can be GL_FRONT_AND_BACK, GL_FRONT, or GL_BACK; mode can be GL_POINT, GL_LINE, or GL_FILL

• void glCullFace(GLenum mode);

  Indicates which polygons should be discarded. The mode is either GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK. To take effect, use glEnable(GL_CULL_FACE).
PolygonMode and CullFace

// for testing polygon mode
    if (cnt % 100 > 50) gl.glPolygonMode(GL.GL_FRONT_AND_BACK, GL.GL_LINE);
    else gl.glPolygonMode(GL.GL_FRONT_AND_BACK, GL.GL_FILL);

// for testing cull face
    if (cnt % 500 > 495) {
        gl.glEnable(GL.GL_CULL_FACE);  gl.glCullFace(GL.GL_FRONT);
    }
    else if (cnt % 500 > 490) {
        gl.glEnable(GL.GL_CULL_FACE);  gl.glCullFace(GL.GL_BACK);
    }
    else {
        gl.glDisable(GL.GL_CULL_FACE);
    }

Copyright @ 2002 by Jim X. Chen:
    jchen@cs.gmu.edu
CIRCLE, CONE, CYLINDER, SPHERE, AND ANIMATION

Circle
Cone
Cylinder
Sphere
Circle

Drawing a circle by subdivision

- $depth=0$, draw 4 triangles. $depth=1$, each triangle is subdivided into two and we draw 8 triangles.

- Consider $v1$, $v2$, and $v12$ as vectors. Then, $v12$ is in the direction of $(v1 + v2) = (v1_x + v2_x, v1_y + v2_y, v1_z + v2_z)$ and $|v1| = |v2| = |v12|$.

- $v12 = \text{normalize}(v1 + v2)$. Normalizing a vector is equivalent to scaling the vector to a unit vector.

- Recursively subdivides $depth$ times and draws $2^{depth}$ triangles.
raise the center of the circle along the $z$ axis

**Example:** \texttt{J2 5 Cone}

```c
// make sure the cone is within the viewing volume
glOrtho(-Width/2, Width/2, -Height/2, Height/2, -Width/2, Width/2);
```
private void subdivideCone(float v1[], float v2[], int depth) {
    float v0[] = {0, 0, 0}; float v12[] = new float[3];

    if (depth==0) {
        gl.glColor3d(v1[0]*v1[0], v1[1]*v1[1], v1[2]*v1[2]);
        drawtriangle(v2, v1, v0); // bottom cover of the cone
        v0[2] = 1; // height of the cone, the tip on z axis
        drawtriangle(v1, v2, v0); // side cover of the cone
        return;
    }

    for (int i = 0; i<3; i++)
        v12[i] = v1[i]+v2[i];
    normalize(v12); subdivideCone(v1, v12, depth-1); subdivideCone(v12, v2, depth-1);
}
draw a circle at \( z=0 \), draw another circle at \( z=1 \).

If we connect the rectangles of the same vertices on the edges of the two circles, we have a cylinder

**Example: J2 6 Cylinder**

```cpp
// make sure the cone is within the viewing volume

glOrtho(-Width/2, Width/2, -Height/2, Height/2, -Width/2, Width/2);
```
• assume equilateral triangle \((v_1, \ v_2, \ v_3)\) on a sphere and \(|v_1|=|v_2|=|v_3|=1\).

• \(v_{12} = \text{normalize}(v_1+v_2)\) is also on the sphere. We can further subdivide the triangle into four equilateral triangles

---

**Example: J2_7_Sphere**

\(a)\) Subdivision  \(b)\) Front view of an octahedron
private void subdivideSphere( float v1[], float v2[], float v3[], long depth) {
    float v12[] = new float[3], v23[] = new float[3], v31[] = new float[3];

    if (depth==0) {
        gl.glColor3f(v1[0]*v1[0], v2[1]*v2[1], v3[2]*v3[2]);
        drawtriangle(v1, v2, v3);
        return;
    }

    for (i = 0; i<3; i++) {
        v12[i] = v1[i]+v2[i];  v23[i] = v2[i]+v3[i];  v31[i] = v3[i]+v1[i];
    }
    normalize(v12); normalize(v23);   normalize(v31);
    subdivideSphere(v1, v12, v31, depth-1);
    subdivideSphere(v2, v23, v12, depth-1);
    subdivideSphere(v3, v31, v23, depth-1);
    subdivideSphere(v12, v23, v31, depth-1);
}
More Hidden-surface Removal

- view point at the origin and the models are in the negative z axis
- render the models in the order of their distances to the view point along z axis from the farthest to the closest
- build up a box (bounding volume) with the faces perpendicular to the x, y, or z axis to bound a 3D model
- compare the minimum and maximum bounds in the z direction between boxes to decide which model should be rendered first
- Using bounding volumes to decide priority of rendering is also known as minmax testing.

Copyright © 2002 by Jim X. Chen: jchen@cs.gmu.edu
Collision Detection

- *collision detection* avoid two models in an animation penetrating each other.

- We can use their bounding volumes to decide their physical distances and collision.

- *bounding volume* can be in a different shape other than a box, such as a sphere.

- We may use multiple *spheres* with different radii to more accurately bound a model, but the collision detection would be more complex.

`J2_11_ConeSolarCollision`
Former Homework

• Implement hidden-surface removal to demonstrate the following that the box is fixed and the triangle is rotating.
Former Homework

Draw a cone, a cylinder, and a sphere that bounce back and forth in a circle. When the models meet, they change their directions of movement

- Must be in double buffer mode
- Must have hidden surface removal

Copyright @ 2002 by Jim X. Chen:
jchen@cs.gmu.edu
HW4: 2012 Fall Class

1. Continue from previous homework;
2. Change the big circle into a rectangle, and you can get rid of the points that move on the circle; (40%)
3. Change the previous bouncing circles into spheres; (40%)
4. Some spheres are filled solid, some are wireframes; (20%)
5. Now you have spheres bouncing in a 2D rectangle.