# Depth Testing & Culling

OpenGL

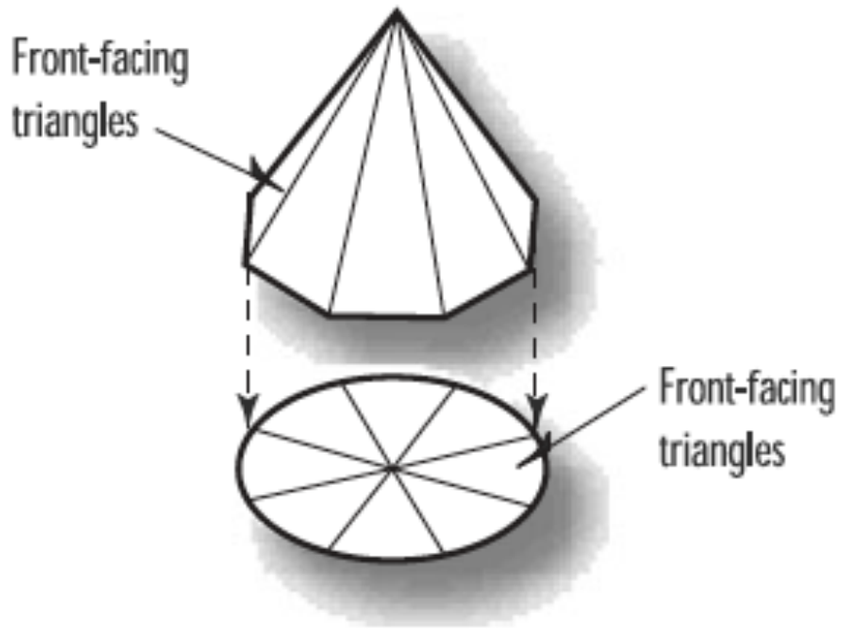
# Learning Outcomes

- Have constructed a solid 3D object
- Understand the Depth Testing model in OpenGL, the use of the Depth Buffer and how to enable and disable it
- Have used Culling, have used it and understand its relationship to winding in OpenGL

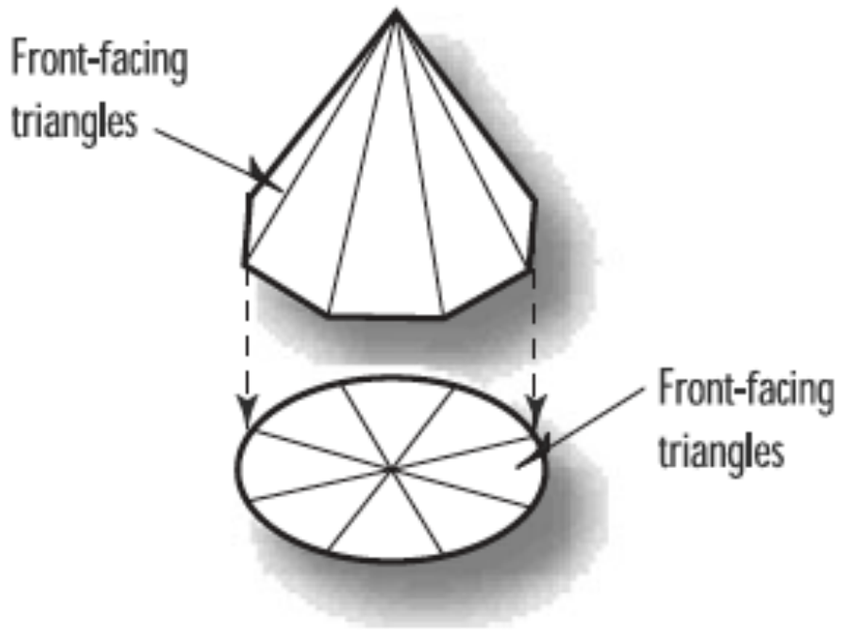
- Objective: use two triangle fans to create a cone.

- The first fan produces the cone shape, using the first vertex as the point of the cone and there remaining vertices as points along a circle farther down the z-axis.

- The second fan forms a circle and lies entirely in the x y plane, making up the bottom surface of the cone.



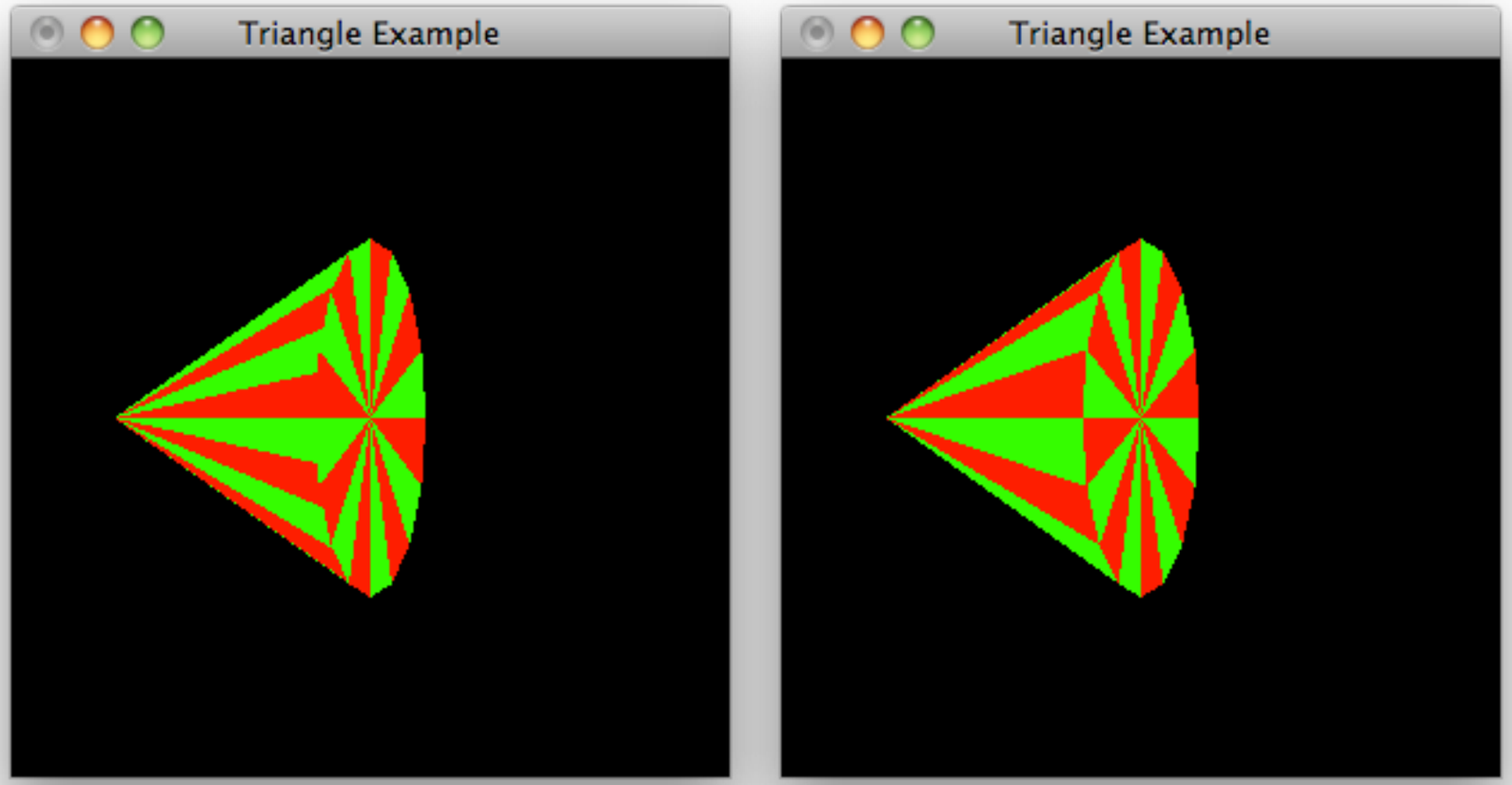
```
void renderScene(void)
{

  glClear(GL_COLOR_BUFFER_BIT);

  drawCircleFan(0,0,75, 50);
  drawCircleFan(0,0,0, 50);

 glutSwapBuffers();
}
```

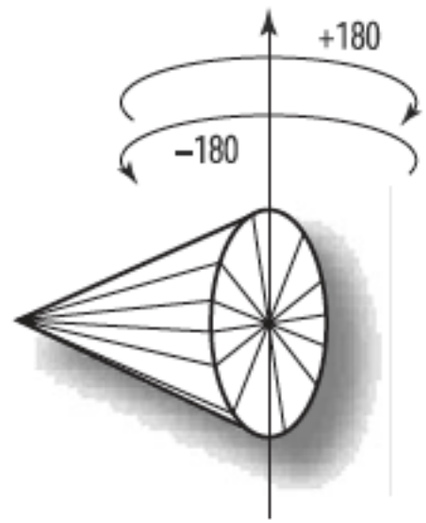


# Hidden Surface Removal

- If we hold down one of the arrow keys to spin the cone around,

- Something unsettling: The cone appears to be swinging back and forth plus and minus 180°, with the bottom of the cone always facing front, but not rotating a full 360°




lab04a_04a

# Problem

- This wobbling happens because the bottom of the cone is drawn after the sides of the cone are drawn.

- No matter how the cone is oriented, the bottom is drawn on top of it, producing the “wobbling” illusion.

- In general if more than one object is drawn and one is in front of the other (from the viewer’s perspective), the last object drawn still appears over the previously drawn object.

# Depth Testing

1. When a pixel is drawn, it is assigned a value (called the z value) that denotes its distance from the viewer's perspective.

2. When another pixel needs to be drawn to that screen location, the new pixel's z value is compared to that of the pixel that is already stored there.

3. If the new pixel's z value is higher, it is closer to the viewer and thus in front of the previous pixel, so the previous pixel is obscured by the new pixel.

4. If the new pixel's z value is lower, it must be behind the existing pixel and thus is not obscured.

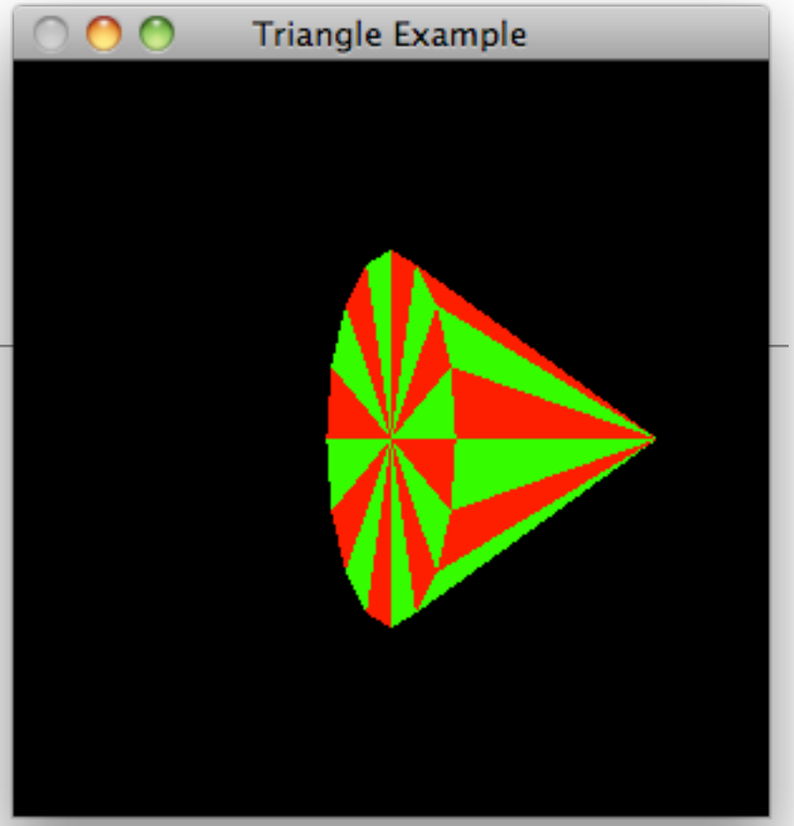
- Depth testing is an effective technique for hidden surface removal, and OpenGL has functions that do this behind the scenes.

- This maneuver is accomplished internally by a depth buffer with storage for a depth value for every pixel on the screen.

# Depth Buffer

- The depth buffer is analogous to the color buffer in that it contains information about the distance of the pixels from the observer.

- This information is used to determine whether any pixels are hidden by pixels closer to the observe



1. Request a depth buffer when you set up your OpenGL window with GLUT

```
int main(int argc, char* argv[])
{
  //...
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  //...
}
```

2. Enable depth testing

```
void retupRC()
{
  //...
  glEnable(GL_DEPTH_TEST);
}
```

3. The Depth buffer must be cleared each time the scene is rendered

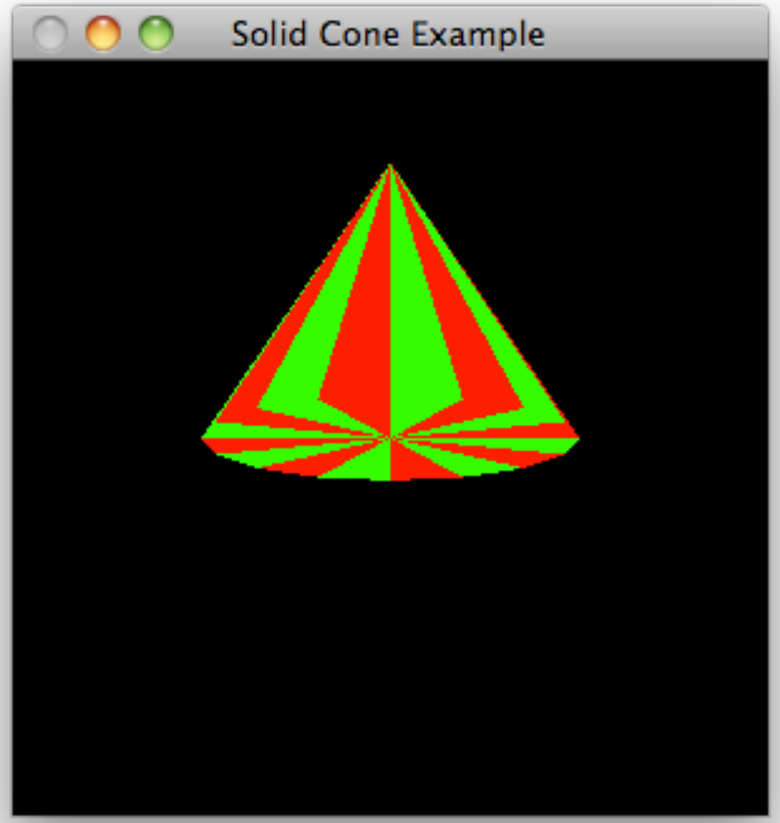
```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  //...
}
```

# Culling: Hiding Surfaces for Performance

- There are obvious visual advantages to not drawing a surface that is obstructed by another.
- Even so, you pay some performance overhead because every pixel drawn must be compared with the previous pixel's z value.
- Often it is known that a surface will never be drawn anyway, so why specify it?
- Culling is the term used to describe the technique of eliminating geometry that we know will never be seen.
- By not sending this geometry to your OpenGL driver and hardware, you can make significant performance improvements.

# Backface culling

- Eliminates the backsides of a surface
- The Cone is a closed surface, and we never see the inside.
- OpenGL is actually (internally) drawing the back sides of the far side of the cone and then the front sides of the polygons facing us.
- Then, by a comparison of z buffer values, the far side of the cone is either overwritten or ignored
- If backface culling is enabled, then the back sides of the polygons will not be drawn at all.



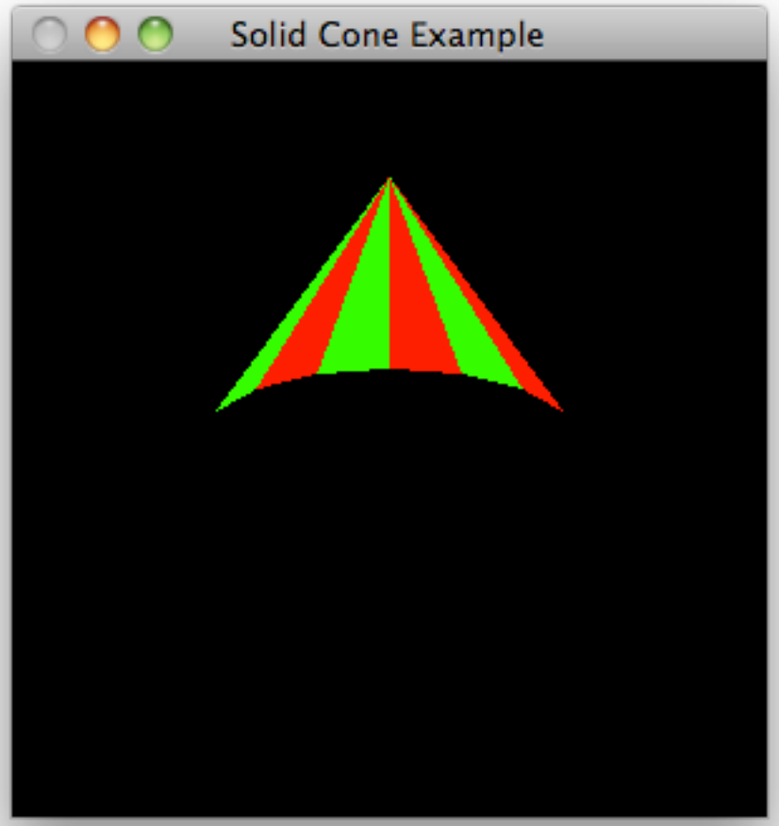

```
void retupRC()
{
  //...
  glEnable(GL_CULL_FACE);
}
```

# Culling and Winding

```
void retupRC()
{
  //...
  glEnable(GL_CULL_FACE);
}
```

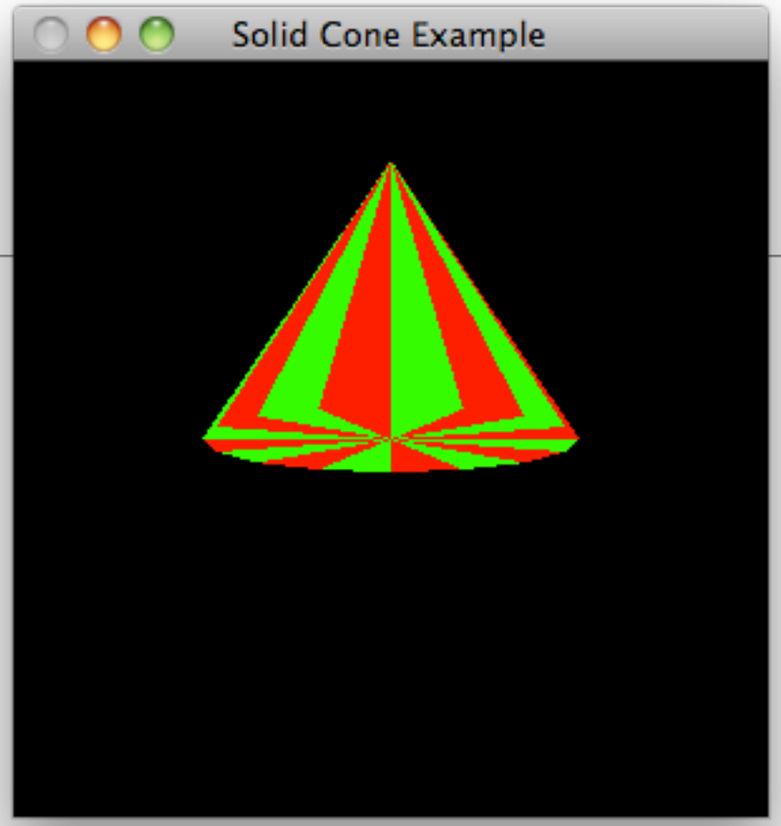
- Care required to avoid culling the wrong face

- Both fans are wound differently, so culling produces incorrect results




lab04a_05a

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  drawCircleFan(0,0,75, 50);
  drawCircleFan(0,0,0, 50);

 glutSwapBuffers();
}
```

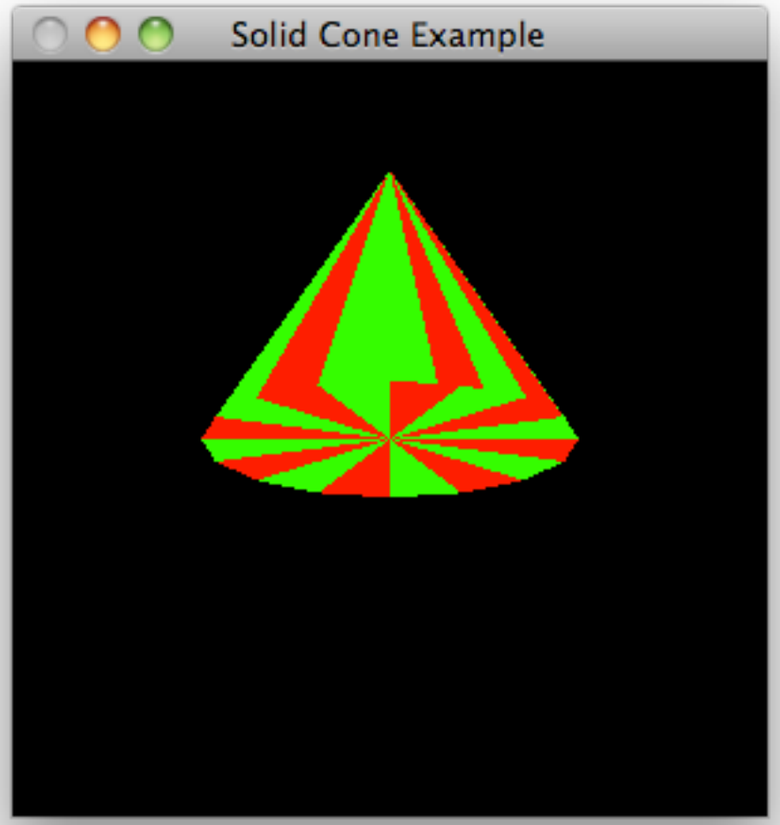



```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glFrontFace(GL_CW);
  drawCircleFan(0,0,75, 50);
  glFrontFace(GL_CCW);
  drawCircleFan(0,0,0, 50);

 glutSwapBuffers();
}
```

lab04a_05b

# No Culling, No Depth Testing

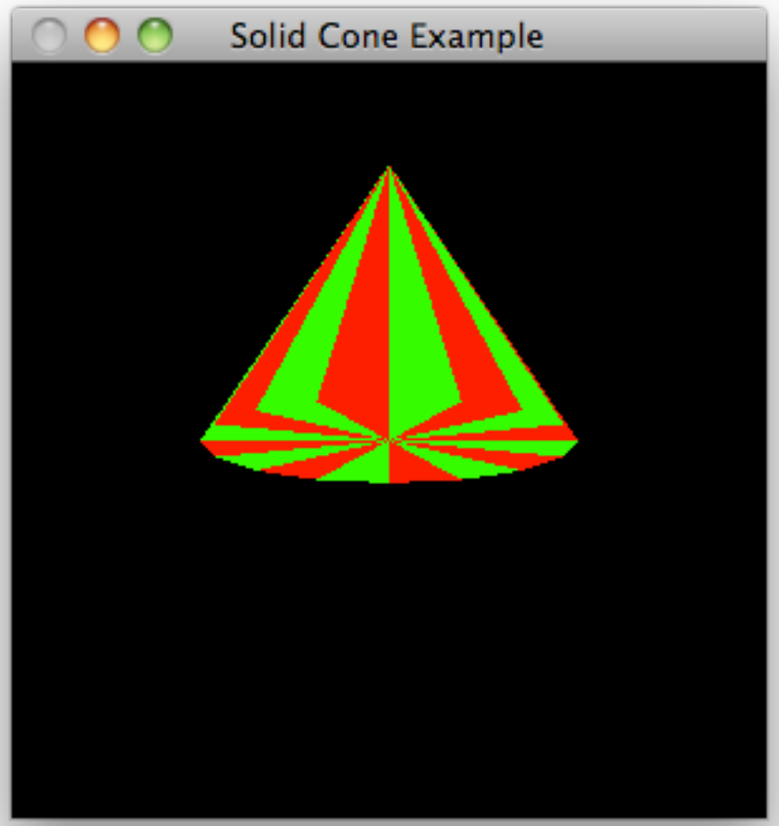
```
void retupRC()
{
  //...
  glPolygonMode(GL_FRONT,GL_FILL);
  glPolygonMode(GL_BACK, GL_FILL);
  glShadeModel(GL_FLAT);
 // glEnable(GL_DEPTH_TEST);
 // glEnable(GL_CULL_FACE);
}
```




lab04a_05c

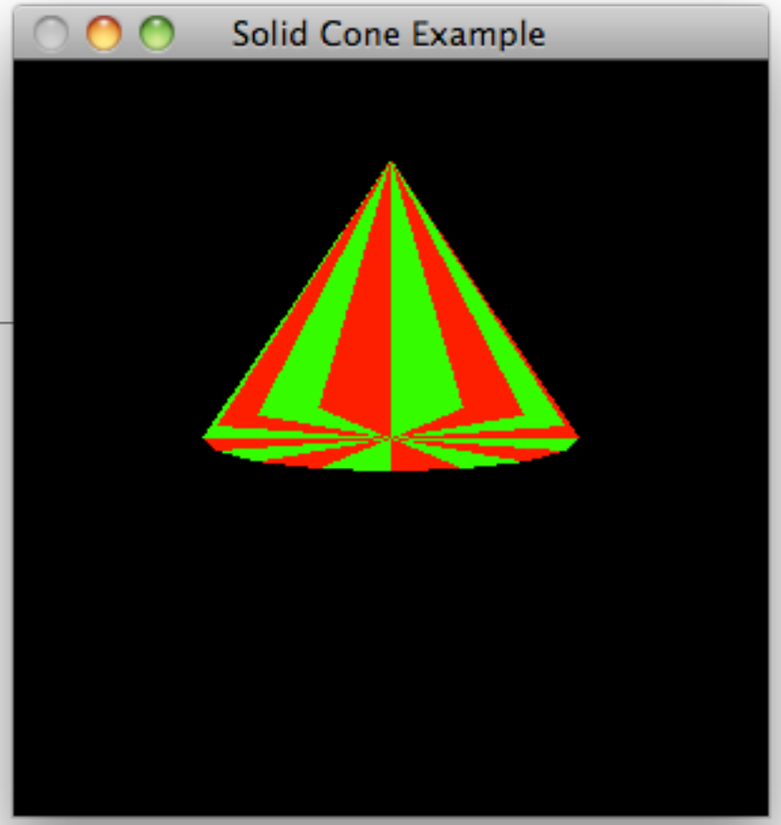
# Culling, No Depth Testing

```
void retupRC()
{
  //...

  glPolygonMode(GL_FRONT,GL_FILL);
  glPolygonMode(GL_BACK, GL_FILL);
  glShadeModel(GL_FLAT);
  //glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);
}
```




lab04a_05d

# Depth Testing, Culling & Winding




- OpenGL uses winding to determine the front and back sides of polygons and that it is important to keep the polygons that define the outside of our objects wound in a consistent direction.
- This consistency is what allows us to tell OpenGL to render only the front, only the back, or both sides of polygons.
- By eliminating the back sides of the polygons, we can drastically reduce the amount of processing necessary to render the image.
- Even though depth testing will eliminate the appearance of the inside of objects, internally OpenGL must take them into account unless we explicitly tell it not to.

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glFrontFace(GL_CW);
  drawCircleFan(0,0,75, 50);
  glFrontFace(GL_CCW);
  drawCircleFan(0,0,0, 50);

  glutSwapBuffers();
}
```

```
void retupRC()
{
  //...
  glPolygonMode(GL_FRONT,GL_FILL);
  glPolygonMode(GL_BACK, GL_FILL);
  glShadeModel(GL_FLAT);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);
}
```