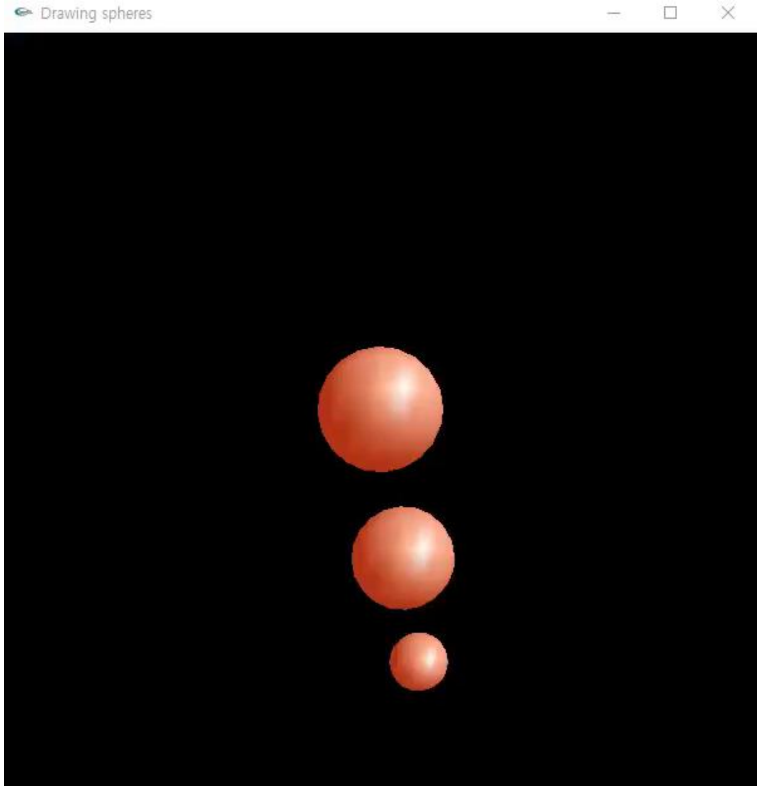
LAB I

Week 07

Seoul National University
Graphics & Media Lab
HyeonSeung Shin

Today's Mission

- Simulate the orbiting spheres
  - Center position and size of each sphere
    - Sphere1: (0,0,0), size = 3
    - Sphere2: (10,0,0), size = 2
    - Sphere3: (15,0,0), size = 1

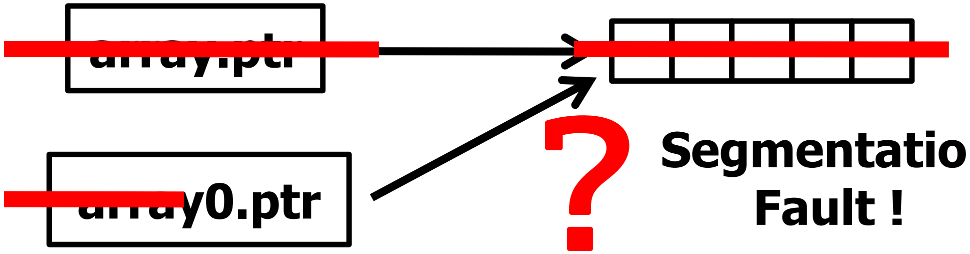


- Technical Guidelines
  - Define sphere class based on the "rule of three"
    - Copy constructor
    - Destructor
    - Assignment operator
  - Draw spheres using glutSolidSphere(), glPushMatrix(), glPopMatrix()
  - Rotate the spheres via idle function

You May Have to Define Your Own Copy Constructor

- Default copy constructor can be problematic.

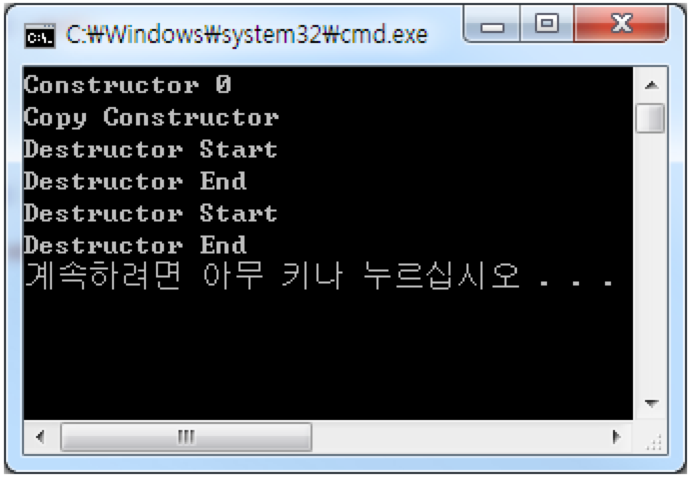
```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }

   int *            ptr;
   std::size_t      size;
};

void f() {
   Array array(5);
   Array array0(array);
}

void main() {
   f();
}
```



Segmentation Fault !

How to Define Your Own Copy Constructor?

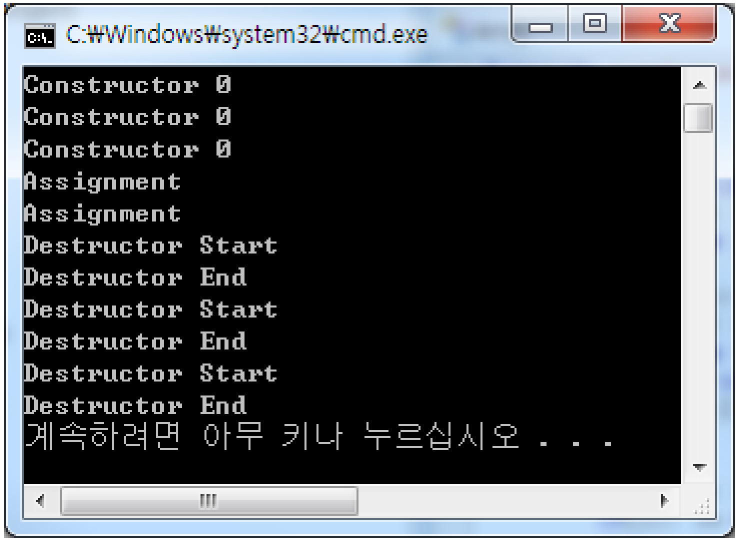
```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   Array(const Array& arr) : size(arr.size) {
      std::cout << "Copy Constructor" << std::endl;
      ptr = new int[size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }

   int *         ptr;
   std::size_t   size;
};

void f() {
   Array array(5);
   Array array0(array);
}

void main() {
   f();
}
```



```
Constructor 0
Copy Constructor
Destructor Start
Destructor End
Destructor Start
Destructor End
계속하려면 아무 키나 누르십시오 . . .
```

How to Define Your Own Assignment Operator?

```
class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   Array(const Array& arr) : size(arr.size) {
      std::cout << "Copy Constructor" << std::endl;
      ptr = new int[size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }
   Array& operator=(const Array& arr) {
      std::cout << "Assignment" << std::endl;
      if(ptr != NULL) delete [] ptr;
      size = arr.size;
      ptr = new int[arr.size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
      return (*this);
   }

   int *          ptr;
   std::size_t    size;
};
void f() {
   Array array(5);
   Array array0(10), array1(10);
   array1 = array0 = array;
}
void main() { f(); }
```



```
Constructor 0
Constructor 0
Constructor 0
Assignment
Assignment
Destructor Start
Destructor End
Destructor Start
Destructor End
Destructor Start
Destructor End
계속하려면 아무 키나 누르십시오 . . .
```

Rule of Three

- In the previous example

```
class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
```

```
   Array(const Array& arr) : size(arr.size) {
      std::cout << "Copy Constructor" << std::endl;
      ptr = new int[size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
   }
```

Copy constructor

```
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }
```

Destructor

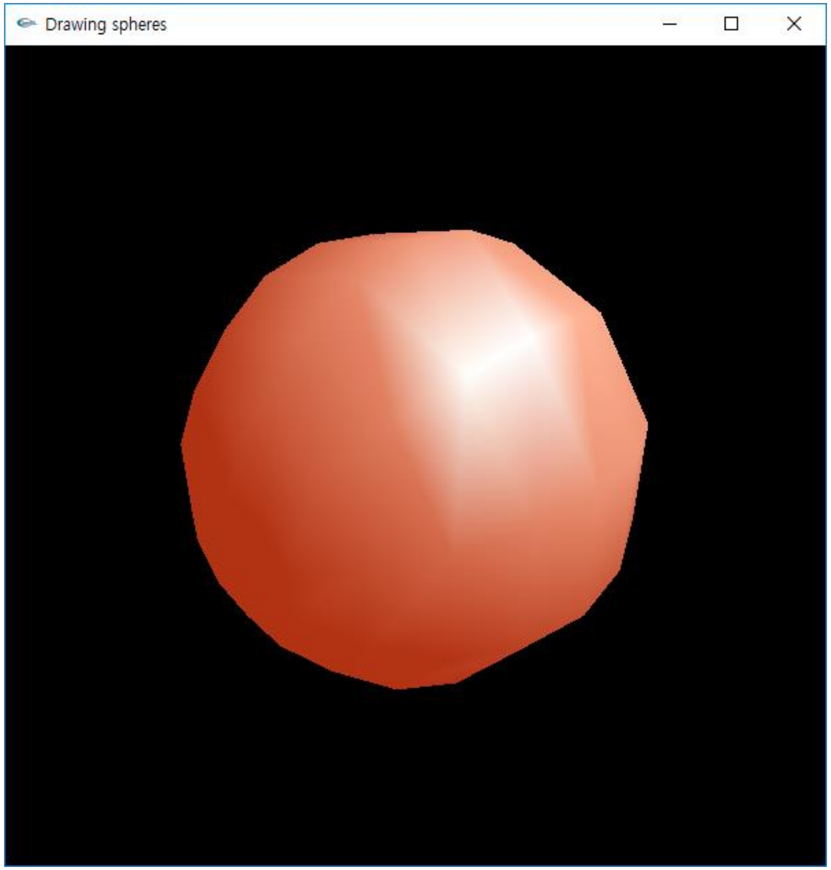
```
   Array& operator=(const Array& arr) {
      std::cout << "Assignment" << std::endl;
      if(ptr != NULL) delete [] ptr;
      size = arr.size;
      ptr = new int[arr.size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
      return (*this);
   }
```

Assignment Operator

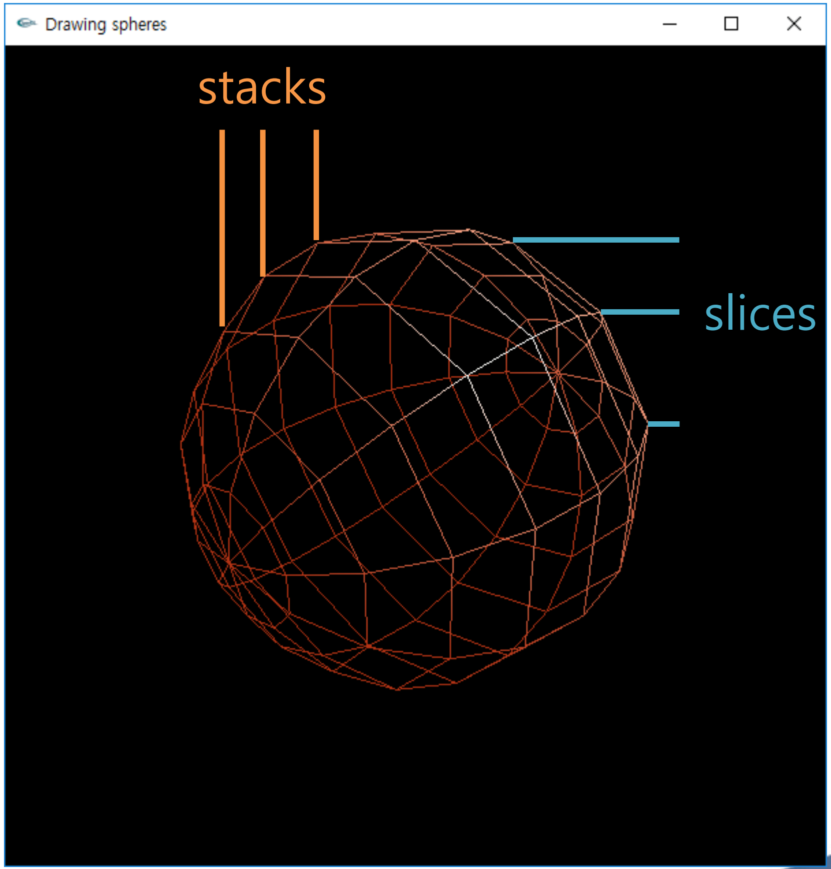
```
   int *           ptr;
   std::size_t    size;
};
```

glutSolidSphere, glutWireSphere

- void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);



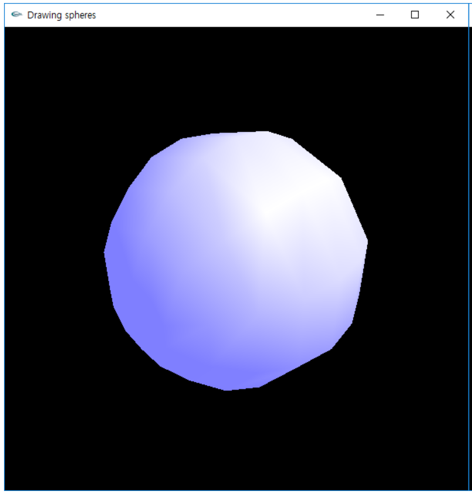
glutSolidSphere(2, 10, 10);



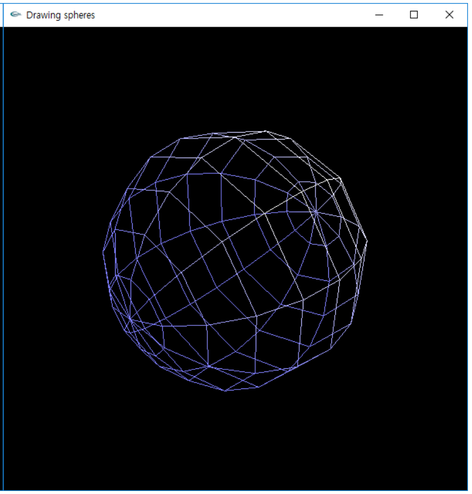
glutWireSphere(2, 10, 10);

Other shapes

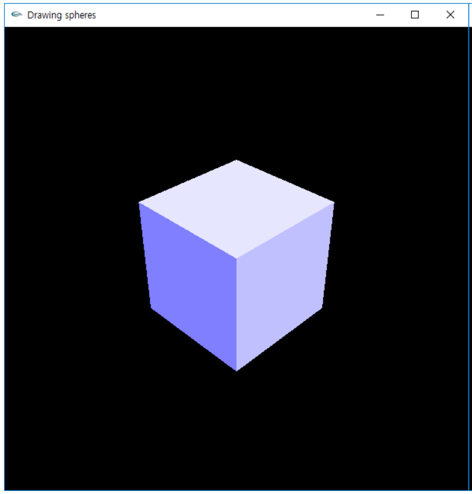
glutSolidSphere(2, 10, 10);



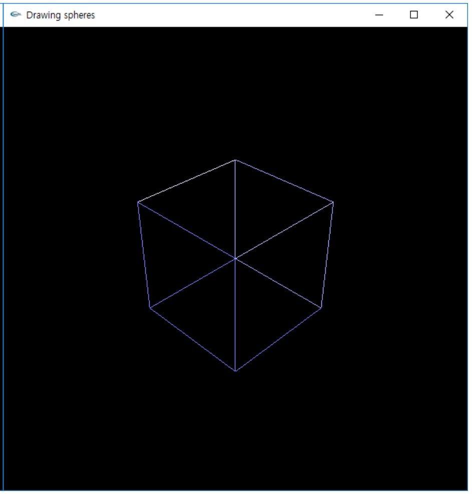
glutWireSphere(2, 10, 10);



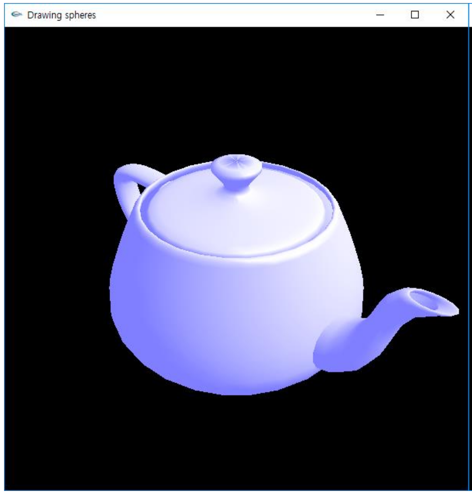
glutSolidCube(2);



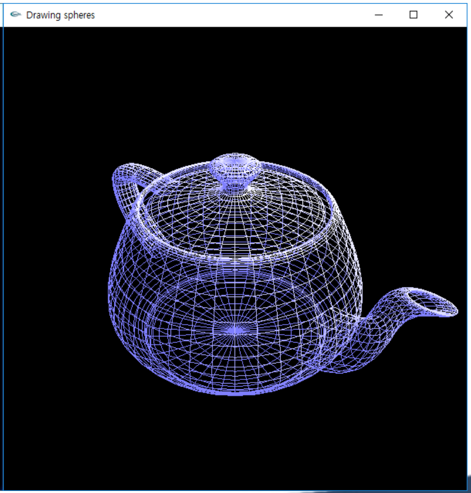
glutWireCube(2);



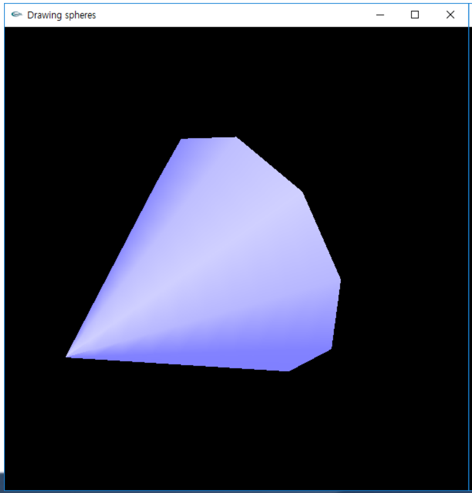
glutSolidTeapot(2);



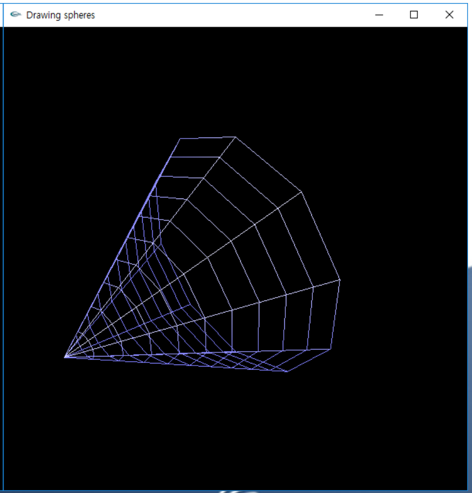
glutWireTeapot(2);



glutSolidCone(2, 3, 10, 10);

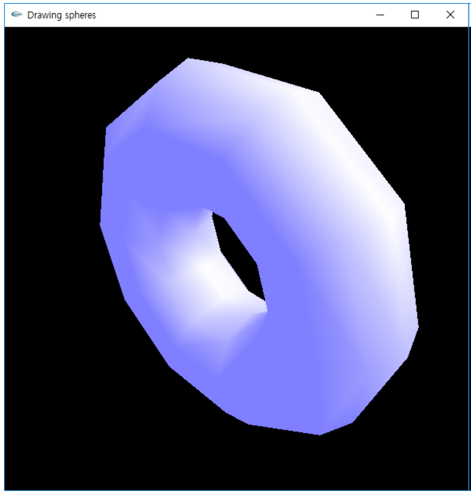


glutWireCone(2, 3, 10, 10);

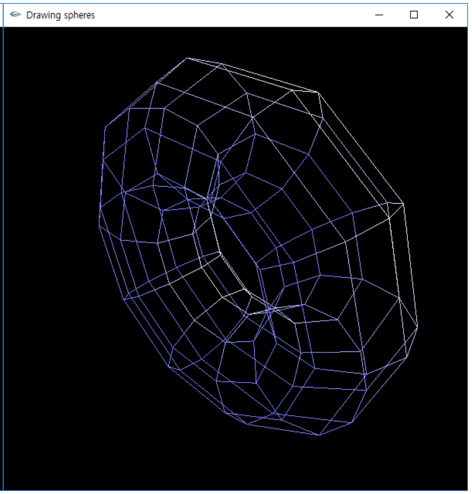


Other shapes

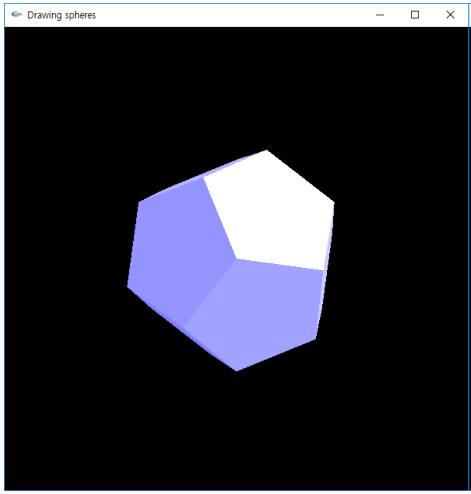
glutSolidTorus(1, 2, 10, 10);



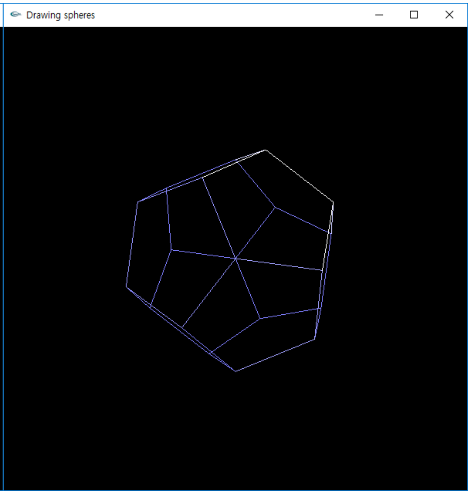
glutWireTorus(1, 2, 10, 10);



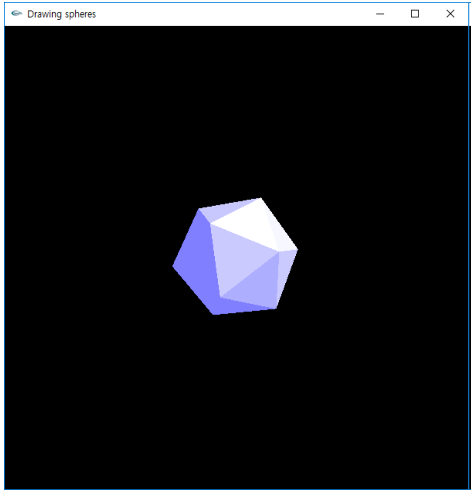
glutSolidDodecahedron();



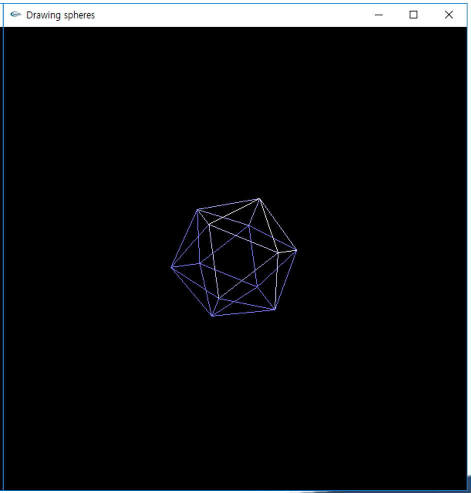
glutWireDodecahedron();



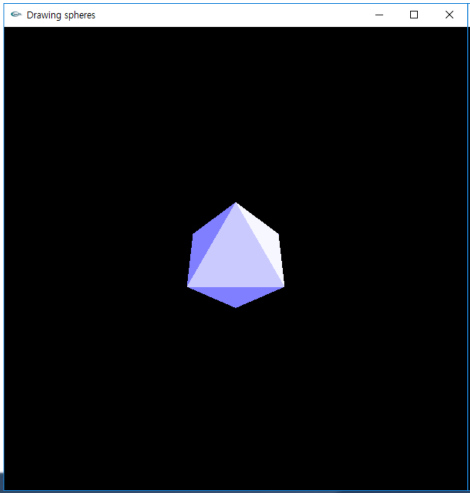
glutSolidIcosahedron();



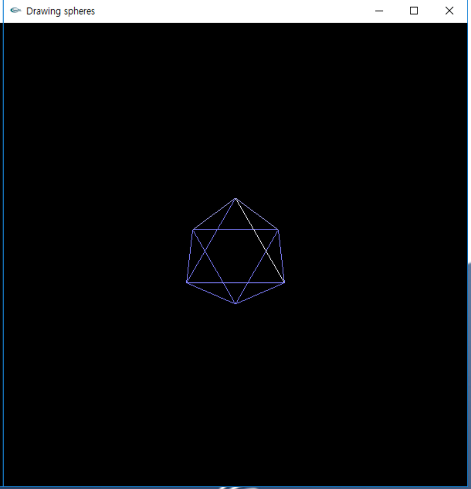
glutWireIcosahedron();



glutSolidOctahedron();

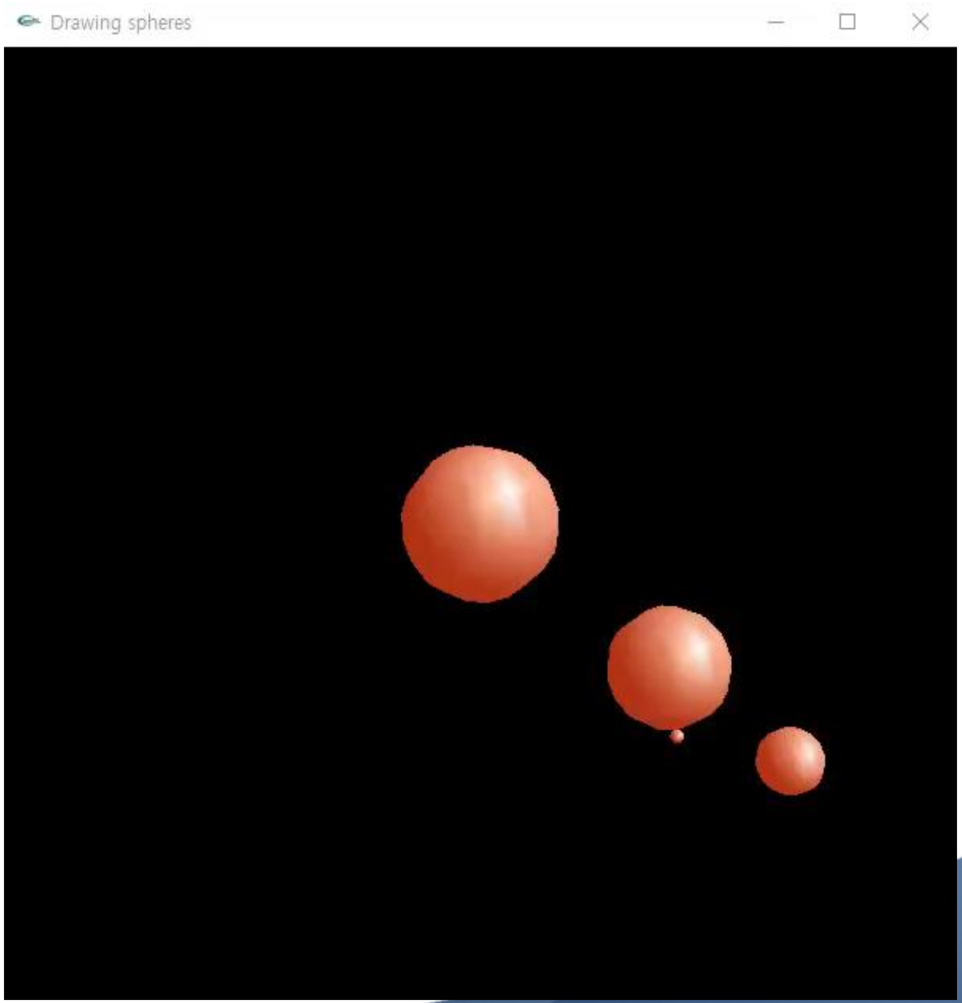


glutWireOctahedron();



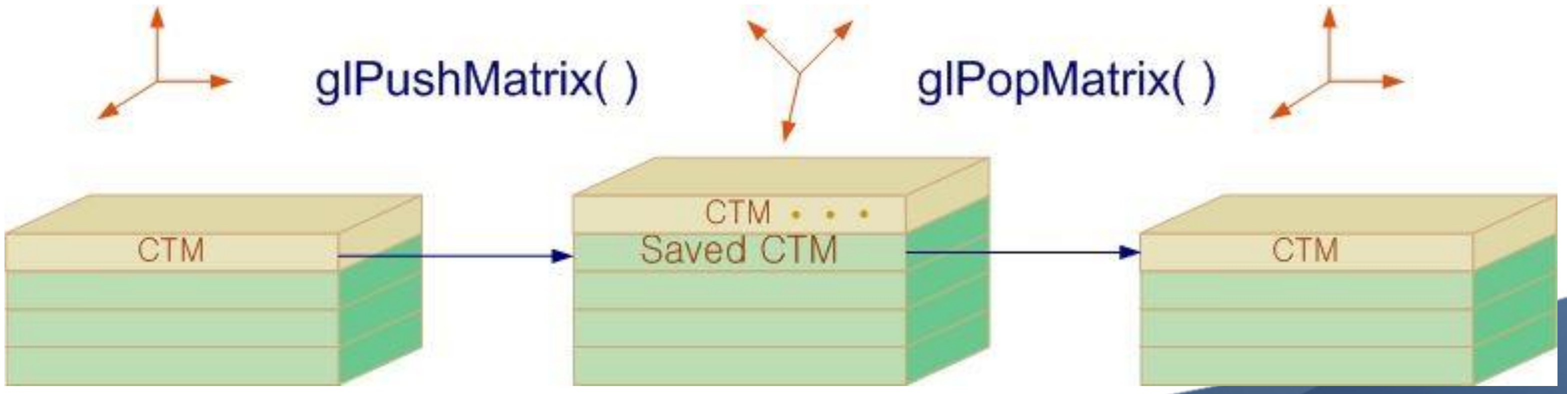
Matrix Stack

```
void renderScene() {
        // Clear Color and Depth Buffers
        glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT);
        …

        glRotatef(angle, 0.0f, 1.0f, 0.0f);
        glutSolidSphere(3, 10, 10);

        glTranslatef(10, 0, 0);
        glutSolidSphere(2, 10, 10);

        glPushMatrix();
        glRotatef(5 * angle, 0.0f, 1.0f, 0.0f);
        glTranslatef(3, 0, 0);
        glutSolidSphere(0.2, 10, 10);
        glPopMatrix();

        glTranslatef(5, 0, 0);
        glutSolidSphere(1, 10, 10);

        glutSwapBuffers();
}
```



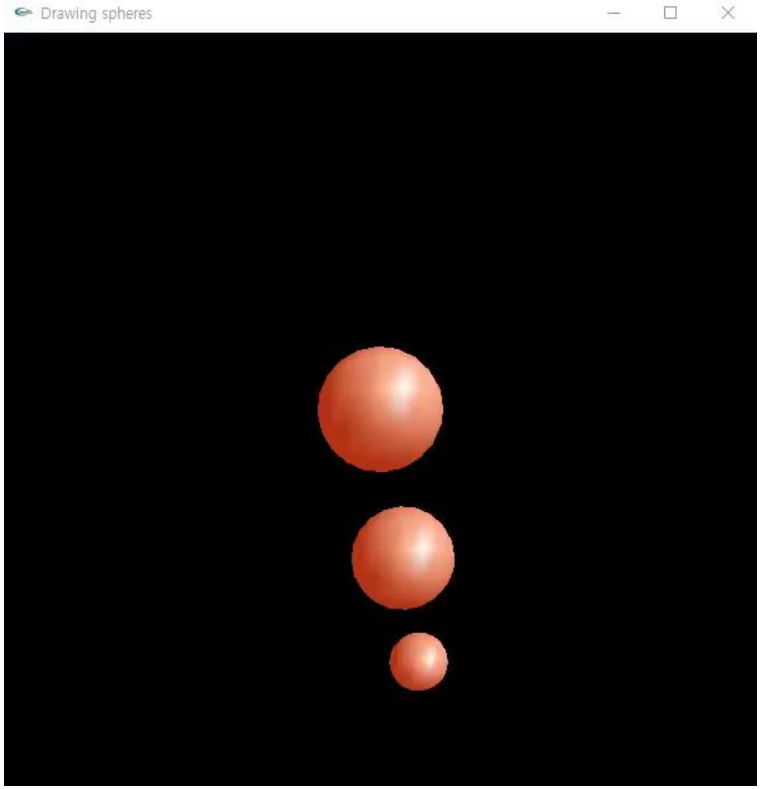
Matrix Stack

- Top matrix in the matrix stack
  - is used as the current transformation matrix
- Pushing and Popping the matrix stack
  - void glPushMatrix( )
    - Push by duplicating the current top matrix
    - Effect: "Remember the current position"
  - void glPopMatrix( )
    - Pop out the top matrix, the second top matrix becoming the new top
    - Effect: "Back to the saved position"



Today's Mission

- Simulate the orbiting spheres
  - Center position and size of each sphere
    - Sphere1: (0,0,0), size = 3
    - Sphere2: (10,0,0), size = 2
    - Sphere3: (15,0,0), size = 1



- Technical Guidelines
  - Define sphere class based on the "rule of three"
    - Copy constructor
    - Destructor
    - Assignment operator
  - Draw spheres using glutSolidSphere(), glPushMatrix(), glPopMatrix()
  - Rotate the spheres via idle function

Given & To Do

- Given
  - Global variable
    - Three spheres
    - Rotation angle

- To Do
  - Separate header and cpp files when declaring and defining class
  - Define sphere class based on the "rule of three"
    - Copy constructor
    - Destructor
    - Assignment operator
  - Draw spheres using glutSolidSphere(), glPushMatrix(), glPopMatrix()
  - Rotate spheres via idle function

To Do (details)

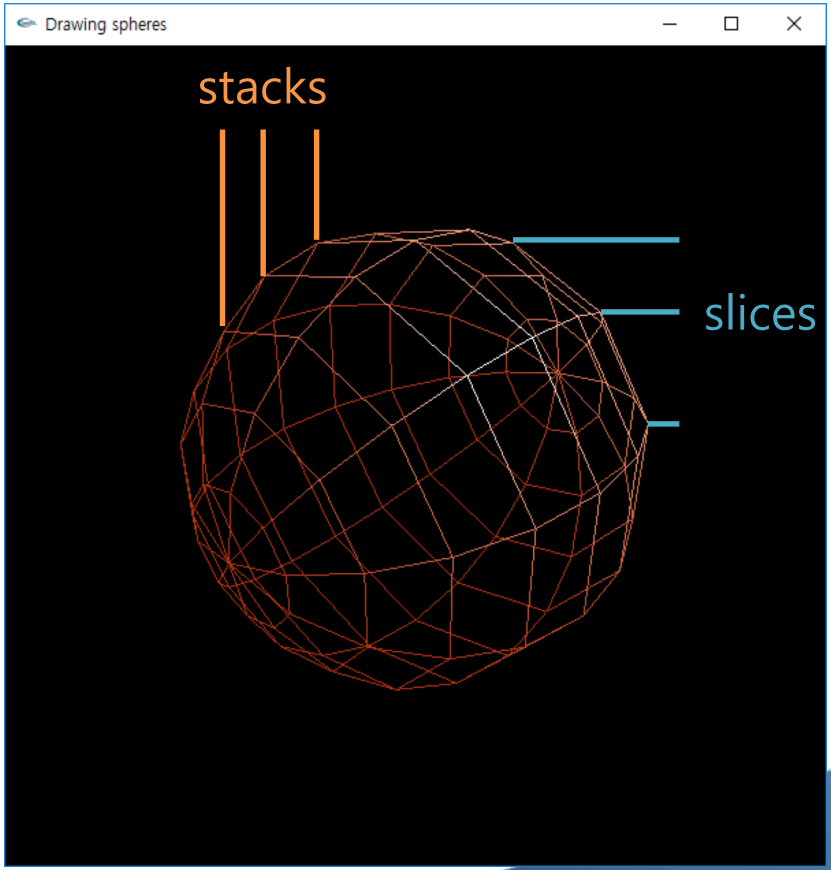
- Center position and size of each sphere
  - Sphere1: (0,0,0), size = 3
  - Sphere2: (10,0,0), size = 2
  - Sphere3: (15,0,0), size = 1

```
Sphere sphere1(0, 0, 0, 3);// Constructor with argument
Sphere sphere2(sphere1);  // Copy constructor
Sphere sphere3;           // Default constructor
float angle = 0;          // Rotation angle

void init() {
        sphere3 = sphere1; // Assignment operator

        /* Set center position and size of each sphere */

        // Light setting
        float light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
        float light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
        float light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        float light_position[] = { 5.0, 5.0, 5.0, 0.0 };
        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);

        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
}
```

To Do (details)

- void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

```
void Sphere::draw() const {
        /* Implement: Draw sphere using glutSolidSphere(),
        glPushMatrix() & glPopMatrix() */

        // Material setting
        float mat_emission[] = { 0.3, 0.0, 0.0, 1.0 };
        float mat_ambient[] = { 1.0, 0.5, 0.2, 1.0 };
        float mat_diffuse[] = { 0.3, 0.5, 0.5, 1.0 };
        float mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        float mat_shininess[] = { 50 };

        glShadeModel(GL_SMOOTH);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

}
```



To Do (details)

- Rotate spheres

```
void idle() {
        /* Implement: Change the rotation angle */
}
```

```
void renderScene() {
        // Clear Color and Depth Buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        // Set the correct perspective.
        gluPerspective(45.0f, 1.0f, 0.1f, 100.0f);

        // Reset transformations
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // Set the camera
        gluLookAt(25.0f, 25.0f, 25.0f,
        0.0f, 0.0f, 0.0f,
        0.0f, 1.0f, 0.0f);

        /* Implement: Rotate spheres */

        sphere1.draw();
        sphere2.draw();
        sphere3.draw();

        glutSwapBuffers();
}
```