OpenGL Programming Part I

Korea Univ.
Computer Graphics Lab.

OpenGL Reference Sites

KUCG

- OpenGL Official Site
  - http://www.opengl.org/
- Khronos Group
  - http://www.khronos.org/
- Nehe Productions
  - http://nehe.gamedev.net/

OpenGL-Related Files

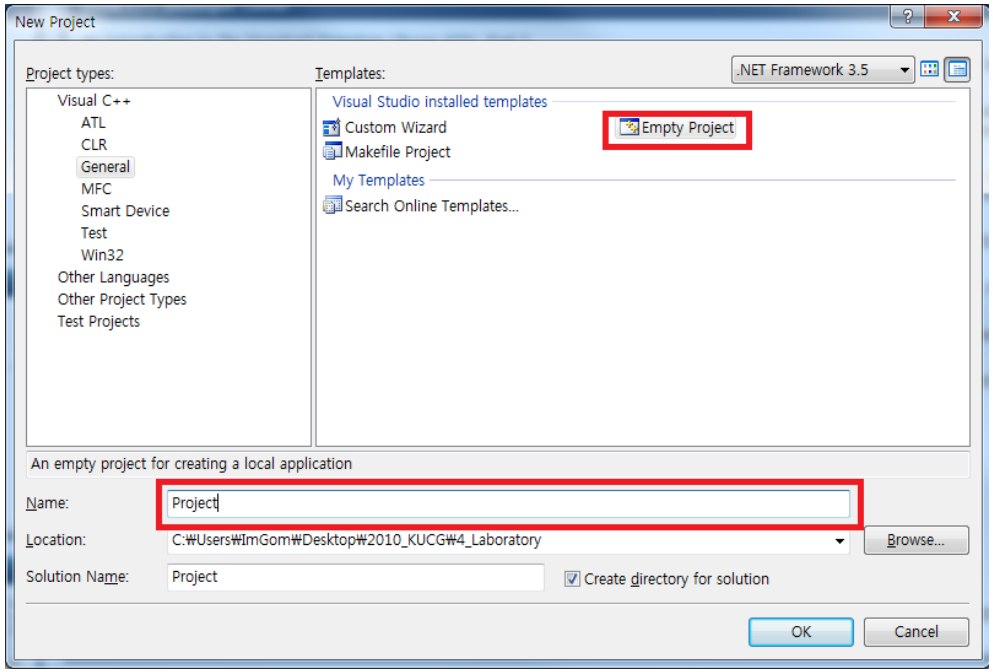
KUCG

- **Header File**
  - gl.h, glaux.h, glu.h, glut.h
  - Microsoft SDKs\v6.0A\Include (VS 2008)
  - or Microsoft SDKs\v7.0A\Include (VS 2010)
- **Static Library**
  - opengl32.lib, glaux.lib, glu32.lib, glut32.lib
  - Microsoft SDKs\v6.0A\Lib (VS 2008)
  - or Microsoft SDKs\v7.0A\Lib (VS 2010)
- **Dynamic Library**
  - opengl32.dll, glu32.dll, glut32.dll
  - C:\WINDOWS\system32

Project Creation [VS 2008] (1/2)

KUCG

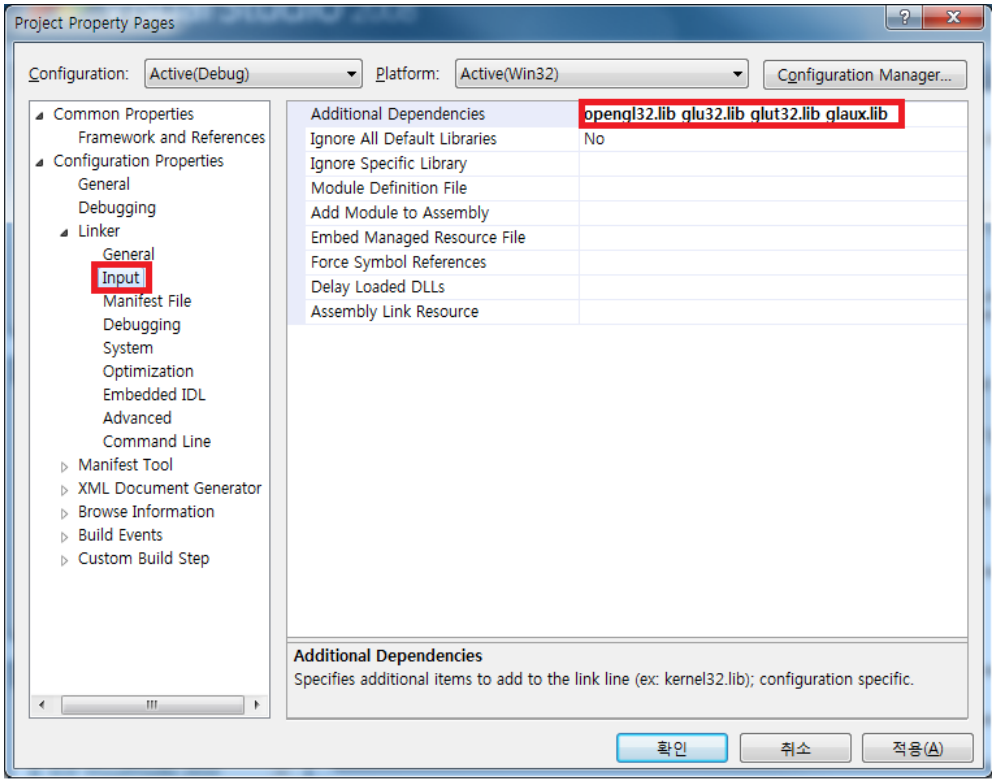
- [File] → [New] (Ctrl+Shift+N)
  - [Project types] → [General]
  - [Templates] → [Empty Project]
  - Enter the project file name



Project Creation [VS 2008] (2/2)

KUCG

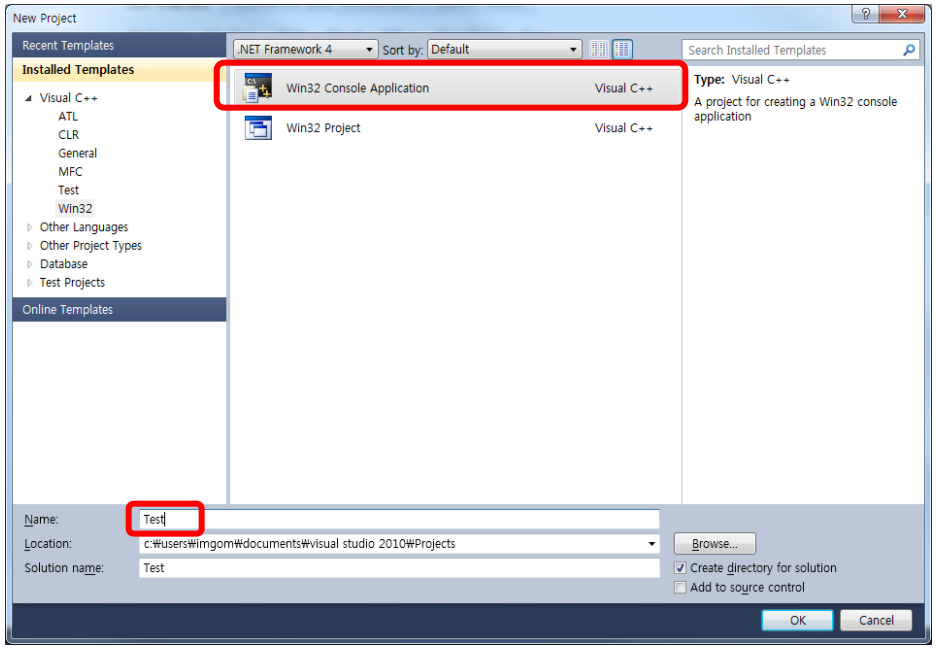
- [Project] → [Settings…] (Alt+F7)
  - [Link] → [Input] → [Additional Dependencies]
    - Type ‘opengl32.lib glu32.lib glut32.lib glaux.lib’

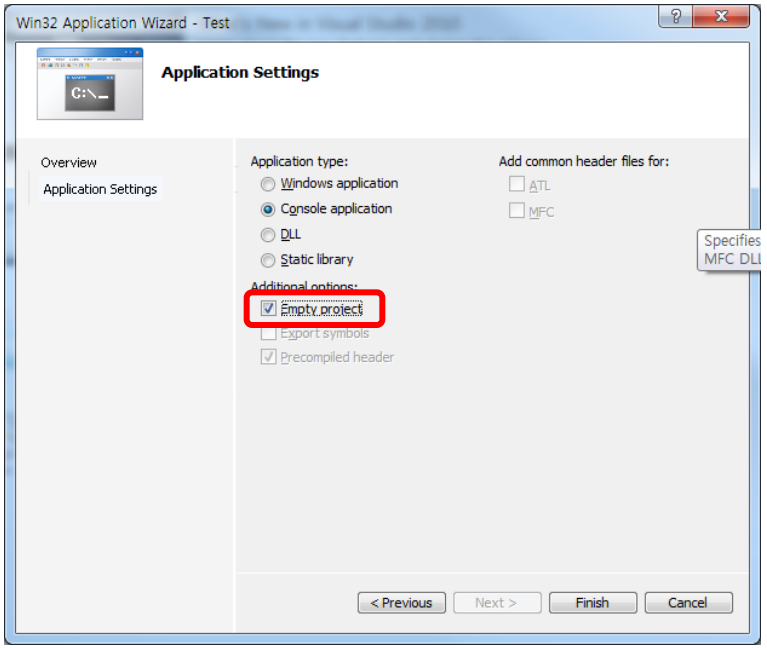


Project Creation [VS 2010] (1/2)

KUCG

- [File] → [New] → [Project] (Ctrl+Shift+N)
  - [Visual C++] → [Win32] → [Win32 Console App]
  - Enter the project name
  - [Empty project]

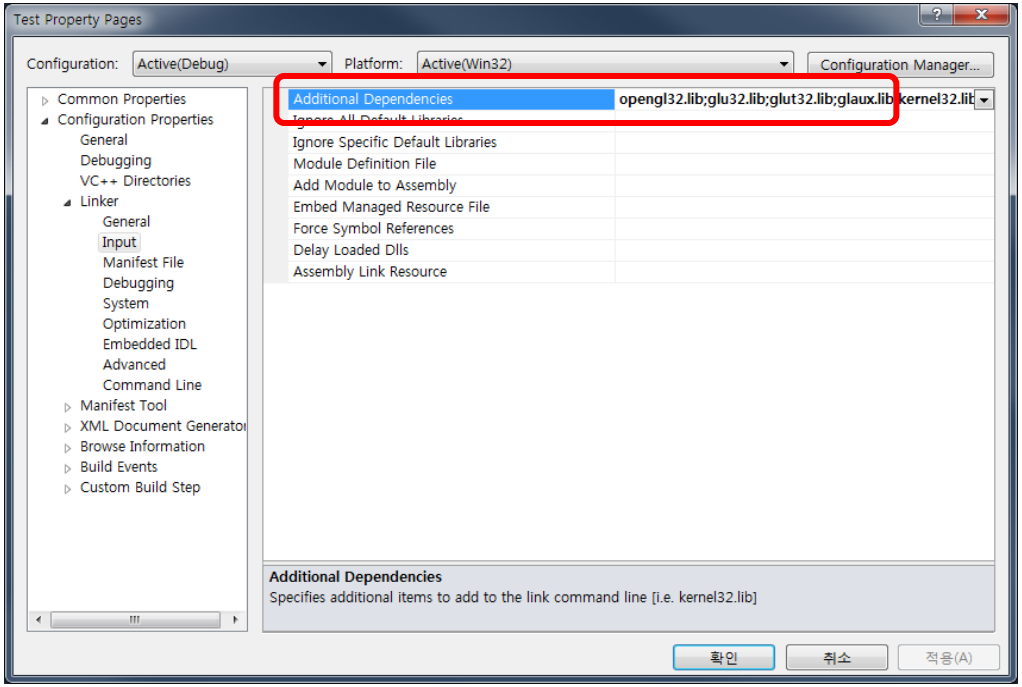




Project Creation [VS 2010] (2/2)

KUCG

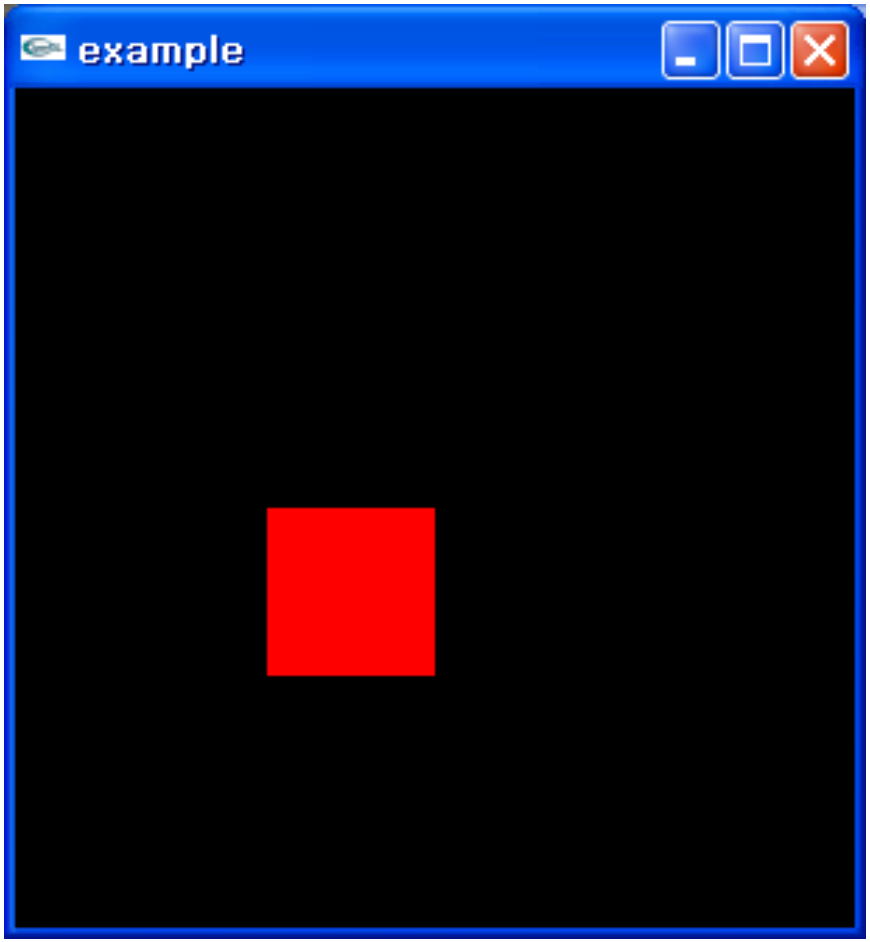
- [Project] → [Properties] (Alt+F7)
  - [Configuration Properties] → [Linker] → [Input]
  - [Additional Dependencies]
    - Type ‘opengl32.lib glu32.lib glut32.lib glaux.lib’



Example

KUCG

```
#include <window.h>
#include <glut.h>

void reshape(int w, int h)
{
    glLoadIdentity();
    glViewport(0,0,w,h);
    glOrtho(0.0, 100.0, 0.0, 100.0, 1.0, -1.0);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glRectf(30.0, 30.0, 50.0, 50.0);
    glutSwapBuffers();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("example");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```



OpenGL Primitives

KUCG

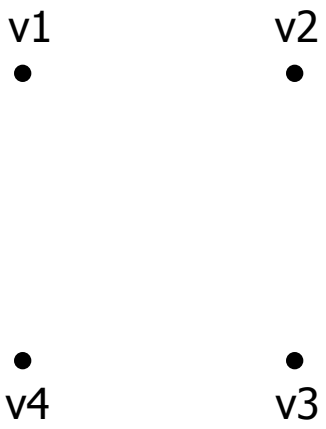
- **Point**
  - GL_POINTS
- **Line**
  - GL_LINES
  - GL_LINE_STRIP
  - GL_LINE_LOOP
- **Polygon**
  - GL_POLYGON
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN
  - GL_QUAD_STRIP
- **Miscellaneous**
  - Cube
  - Tetrahedron
  - Icosahedron
  - Sphere
  - Torus
  - Cone

Point

KUCG

- **GL_POINTS**

```
glBegin(GL_POINTS);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
glEnd();
```

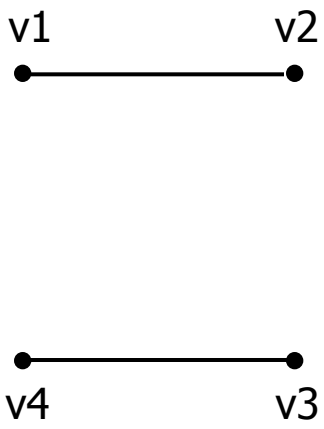


Line (1/3)

KUCG

- **GL_LINES**

```
glBegin(GL_LINES);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
glEnd();
```

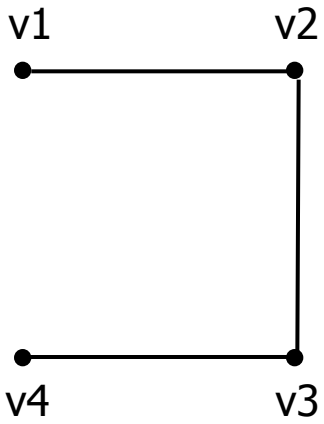


Line (2/3)

KUCG

- **GL_LINE_STRIP**

```
glBegin(GL_LINE_STRIP);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
glEnd();
```

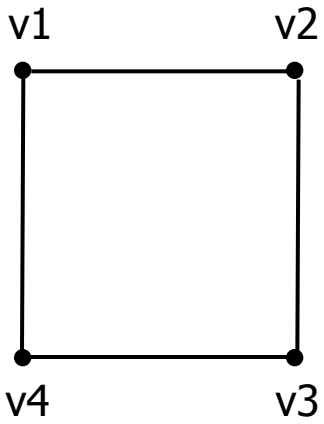


Line (3/3)

KUCG

- **GL_LINE_LOOP**

```
glBegin(GL_LINE_LOOP);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
glEnd();
```

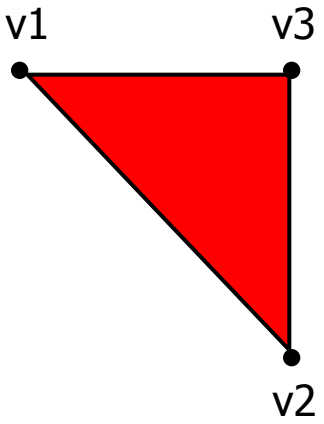


Triangle (1/3)

KUCG

- **GL_TRIANGLES**

```
glBegin(GL_TRIANGLES);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
glEnd();
```

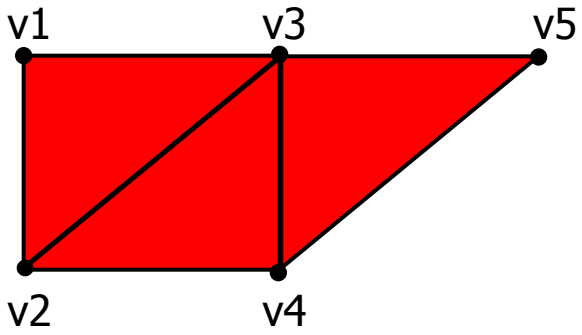


Triangle (2/3)

KUCG

- **GL_TRIANGLE_STRIP**

```
glBegin(GL_TRIANGLE_STRIP);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
   glVertex3f(v5x, v5y, v5z);
glEnd();
```

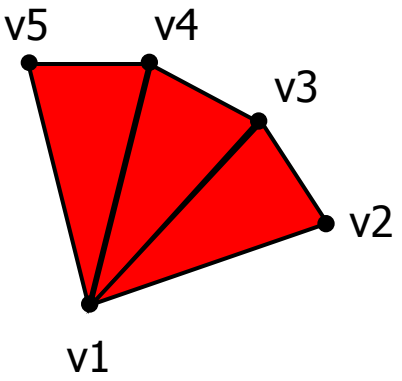


Triangle (3/3)

KUCG

- **GL_TRIANGLE_FAN**

```
glBegin(GL_TRIANGLE_FAN);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
   glVertex3f(v5x, v5y, v5z);
glEnd();
```

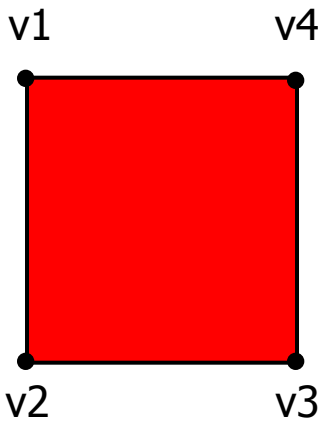


Quadrilateral (1/2)

KUCG

- **GL_QUADS**

```
glBegin(GL_QUADS);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
glEnd();
```

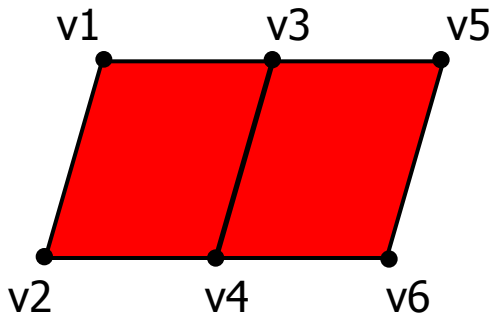


Quadrilateral (2/2)

KUCG

- **GL_QUAD_STRIP**

```
glBegin(GL_QUAD_STRIP);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
   glVertex3f(v5x, v5y, v5z);
   glVertex3f(v6x, v6y, v6z);
glEnd();
```

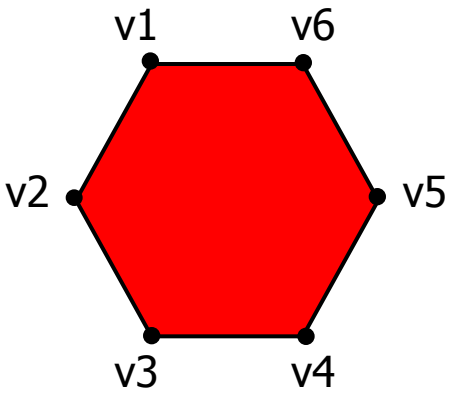


Polygon

KUCG

- **GL_POLYGON**

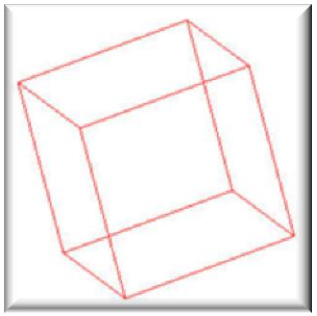
```
glBegin(GL_POLYGON);
   glVertex3f(v1x, v1y, v1z);
   glVertex3f(v2x, v2y, v2z);
   glVertex3f(v3x, v3y, v3z);
   glVertex3f(v4x, v4y, v4z);
   glVertex3f(v5x, v5y, v5z);
   glVertex3f(v6x, v6y, v6z);
glEnd();
```



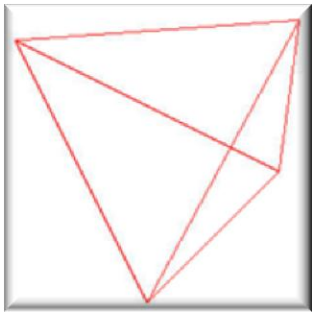
Miscellaneous (1/3)

KUCG

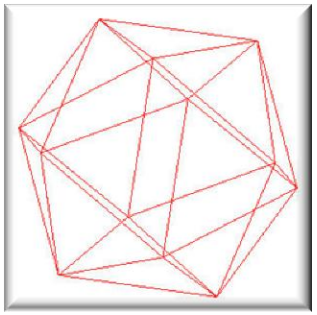
- Cube
  - void glutSolidCube(GLdouble size)
  - void glutWireCube(GLdouble size)



- Tetrahedron
  - void glutSolidTetrahedron(void)
  - void glutWireTetrahedron(void)



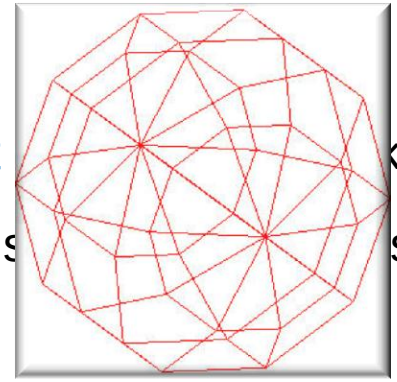
- Icosahedron
  - void glutSolidIcosahedron(void)
  - void glutWireIcosahedron(void)



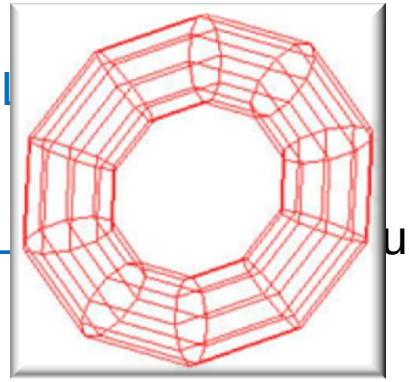
Miscellaneous (2/3)

KUCG

- Sphere
  - void glutSolidSphere(GLdouble radius, GLint ks)
  - void glutWireSphere(GLdouble radius, GLint s s)



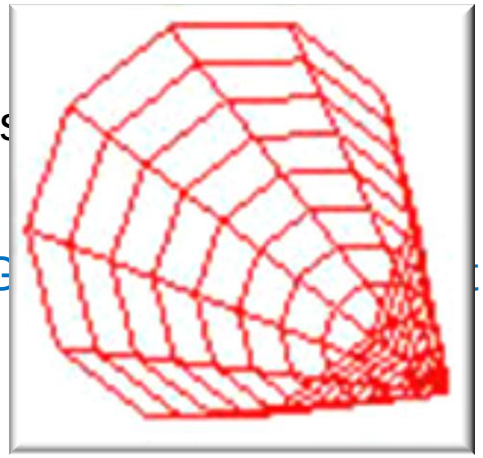
- Torus
  - void glutSolidTorus(GLdouble innerRadius, GL outerRadius, GLint nsides, GLint rings)
  - void glutWireTorus(GLdouble innerRadius, GL us, GLint nsides, GLint rings)



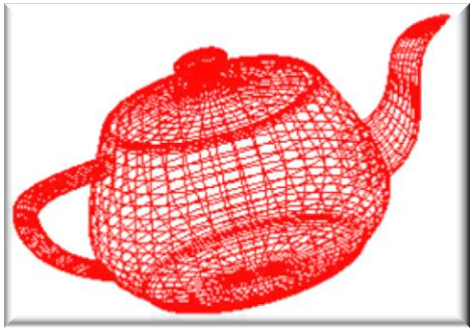
Miscellaneous (3/3)

KUCG

- Cone
  - void glutSolidCone(GLdouble base_radius, GLint slices, GLint stacks)
  - void glutWireCone(GLdouble base_radius, GLint slices, GLint stacks)



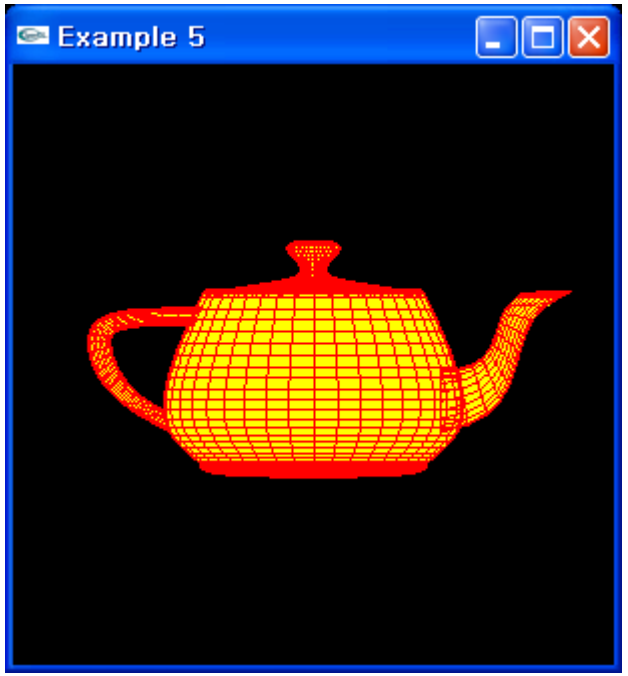
- Teapot
  - void glutSolidTeapot(GLdouble size)
  - void glutWireTeapot(GLdouble size)



Teapot Example

KUCG

```
#include <glut.h>

void myDisplay( )
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 0.0);
    glutSolidTeapot(0.5);
    glColor3f(1.0, 0.0, 0.0);
    glutWireTeapot(0.5);

    glFlush( );
}

void main(int argc, char** argv)
{
    glutCreateWindow("Example 5");
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glutDisplayFunc(myDisplay);
    glutMainLoop( );
}
```



Basic Functions (1/2)

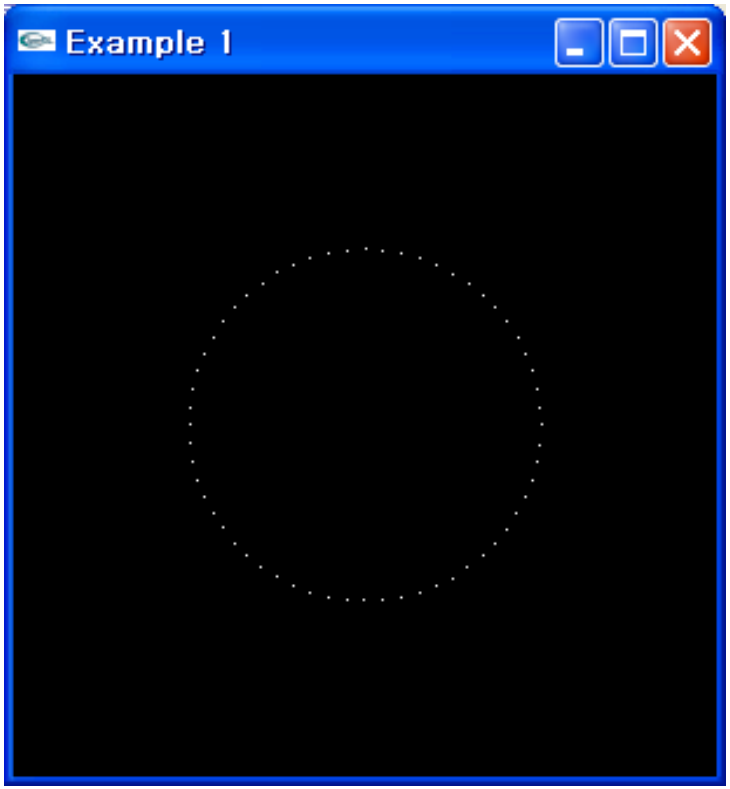
KUCG

- glPointSize(GLfloat size)
  - glGetFloatv(GL_POINT_SIZE_RANGE)
    - Returns the range of the point size that the hardware supports
- glLineWidth(GLfloat width)
  - glGetFloatv(GL_LINE_WIDTH_RANGE)
    - Returns the range of the line width that the hardware supports

Point Example

KUCG

```
#include <glut.h>
#include <math.h>
#define Pi 3.14

void myDisplay( )
{
    GLfloat size[2];
    GLfloat angle;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glGetFloatv(GL_POINT_SIZE_RANGE, size);
    glPointSize(size[0]);
    glBegin(GL_POINTS);
    for(angle = 0.0; angle <= 2.0 * Pi; angle += Pi/ 30.0)
        glVertex3f(0.5 * cos(angle), 0.5 * sin(angle), 0.0);
    glEnd( );
    glFlush( );
}

void main(int argc, char** argv)
{
    glutCreateWindow("Example 1");
    glutDisplayFunc(myDisplay);
    glutMainLoop( );
}
```



Basic Functions (2/2)

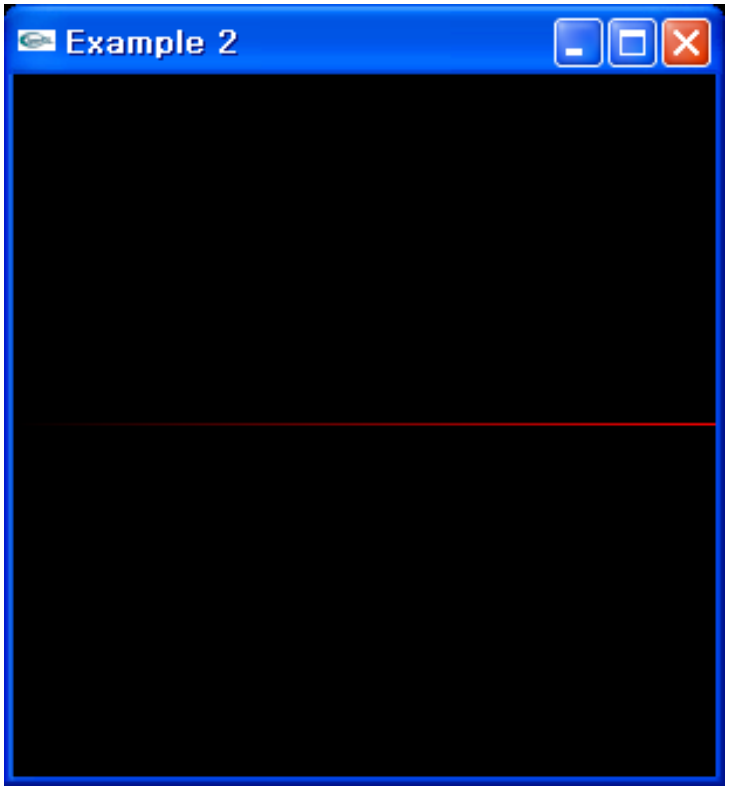
KUCG

- **glShadeModel(mode)**
  - Sets the polygon filling method
  - mode
    - GL_FLAT
      - By only one color
    - GL_SMOOTH
      - By the weighted averaging the colors of member vertices (gradation)
      - Default value

Line Example

KUCG

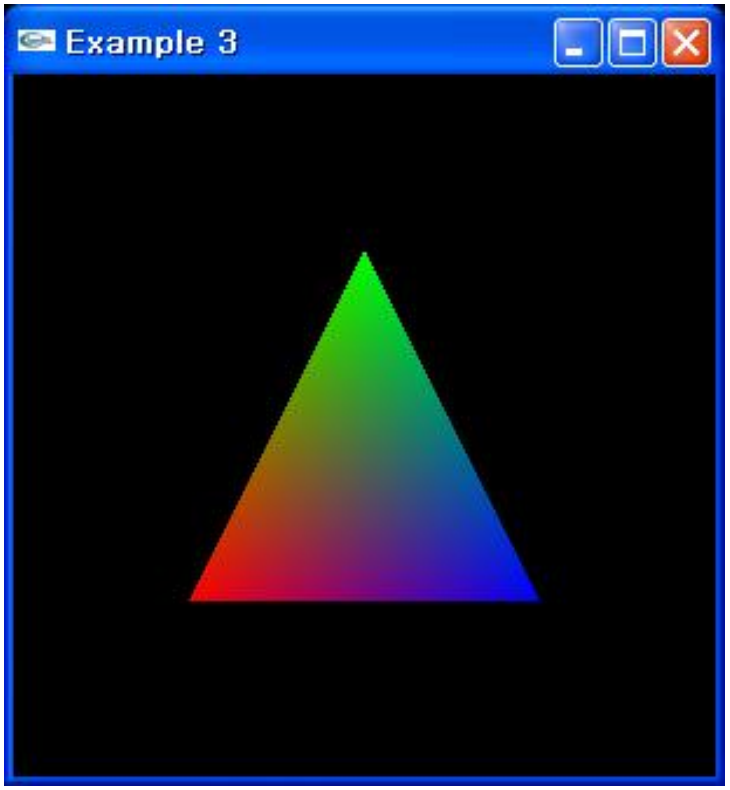
```
#include <glut.h>

void myDisplay( )
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
        glLineWidth(5.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f(0.0, 0.0, 0.0);
        glVertex3f(-1.0, 0.0, 0.0);
    glEnd( );
    glFlush( );
}

void main(int argc, char** argv)
{
    glutCreateWindow("Example 2");
    glutDisplayFunc(myDisplay);
    glutMainLoop( );
}
```



Triangle Example

KUCG

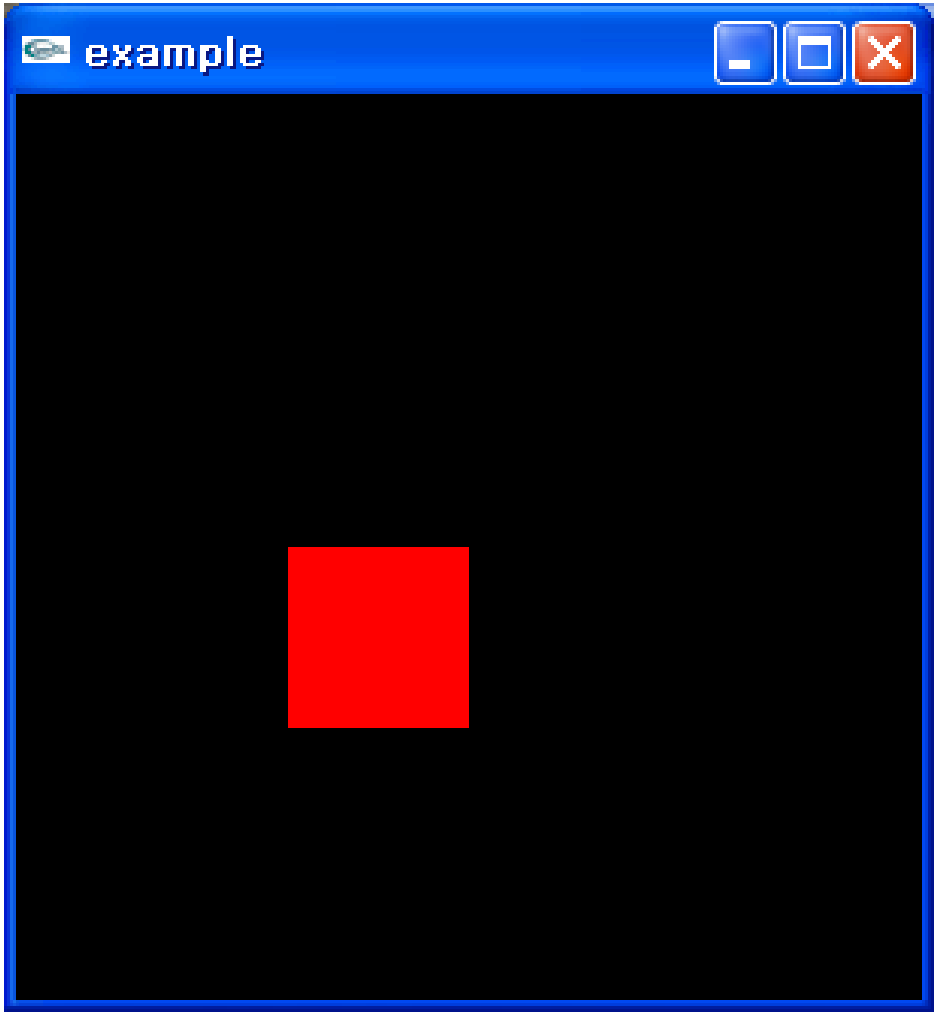
```
#include <glut.h>

void myDisplay( )
{
    glClear(GL_COLOR_BUFFER_BIT);
    glShadeModel(GL_SMOOTH);
    glBegin(GL_TRIANGLES);
        glColor3f(0.0, 1.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(-0.5, -0.5, 0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.5, -0.5, 0.0);
    glEnd( );
    glFlush( );
}

void main(int argc, char** argv)
{
    glutCreateWindow("Example 3");
    glutDisplayFunc(myDisplay);
    glutMainLoop( );
}
```



Rectangle Example : Revisited

KUCG

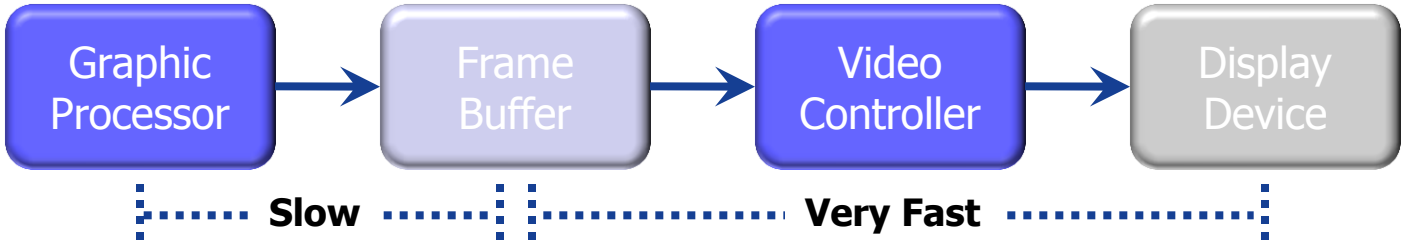
```
#include <window.h>
#include <glut.h>

void reshape(int w, int h)
{
    glLoadIdentity();
    glViewport(0,0,w,h);
    glOrtho(0.0, 100.0, 0.0, 100.0, 1.0, -1.0);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glRectf(30.0, 30.0, 50.0, 50.0);
    glutSwapBuffers();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("example");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```



'Single vs. Double Buffering'
glFlush() vs. glutSwapBuffers()

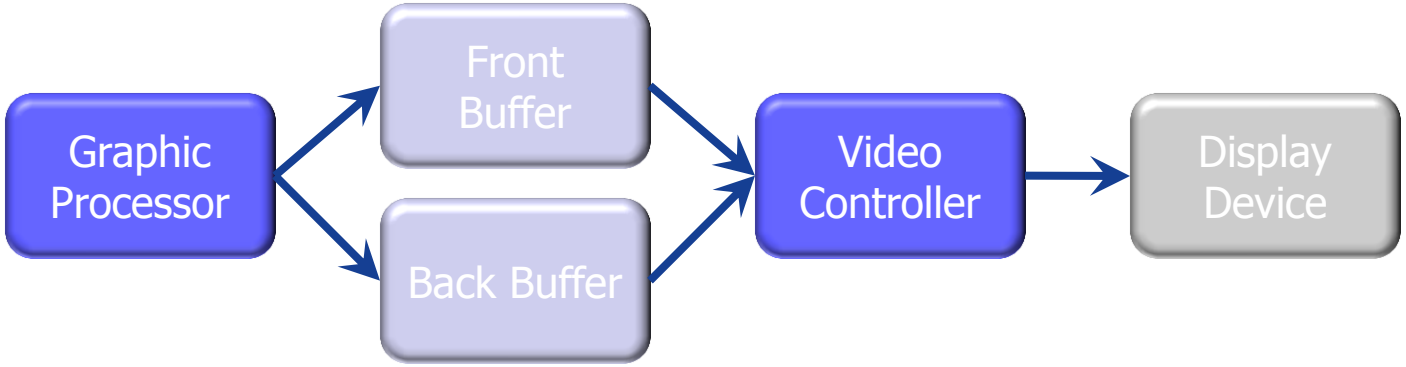
KUCG

- Single Buffering (GLUT_SINGLE, Default)



- Double Buffering (GLUT_DOUBLE)

  cf.) triple buffering



Single Buffering vs. Double Buffering : Assumption & Environment

KUCG

- Assumption
  - Time to write a frame (graphic processor, GPU)
    - 10ms
  - Time to read a frame (video controller, CTRL)
    - 3ms
  - No delay for switching read/write modes
  - Can’t read & write the frame buffer at the same time
    - Mutual exclusion
- Compare the time to display 3 frames

Single Buffering vs. Double Buffering : Single Buffering

KUCG

- Step 1 (0ms) : 10ms
  - GPU
    - Write frame #1 into the frame buffer
- Step 2 (10ms) : 3ms
  - CTRL
    - Read frame #1 from the frame buffer
    - Shoot frame #1 to the display
- Step 3 (13ms) : 10ms
  - GPU
    - Write frame #2 into the frame buffer
- Step 4 (23ms) : 3ms
  - CTRL
    - Read frame #2 from the frame buffer
    - Shoot frame #2 to the display
- Step 5 (26ms) : 10ms
  - GPU
    - Write frame #3 into the frame buffer
- Step 6 (36ms) : 3ms
  - CTRL
    - Read frame #3 from the frame buffer
    - Shoot frame #3 to the display
- Complete (39ms)
  - **Avg. 13ms/frame**

Single Buffering vs. Double Buffering : Double Buffering

KUCG

- Step 1 (0ms) : 10ms
  - GPU
    - Write frame #1 into the front buffer
- Step 2 (10ms) : 3ms
  - CTRL
    - Read frame #1 from the front buffer
    - Shoot frame #1 to the display
  - GPU
    - Write frame #2 (30%) into the back buffer
- Step 3 (13ms) : 7ms
  - GPU
    - Write frame #2 (70%) into the back buffer
- Step 4 (20ms) : 3ms
  - CTRL
    - Read frame #2 from the back buffer
    - Shoot frame #2 to the display
  - GPU
    - Write frame #3 (30%) into the back buffer
- Step 5 (23ms) : 7ms
  - GPU
    - Write frame #3 into the back buffer
- Step 6 (30ms) : 3ms
  - CTRL
    - Read frame #3 from the back buffer
    - Shoot frame #3 to the display
- Complete (33ms)
  - **Avg. 11ms/frame**