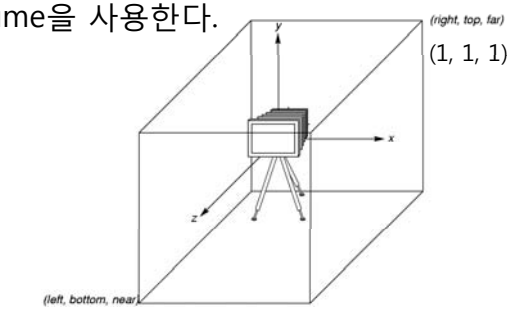
Graphics Programming

321190
2011년 봄학기
3/17/2011
박경신

OpenGL Camera

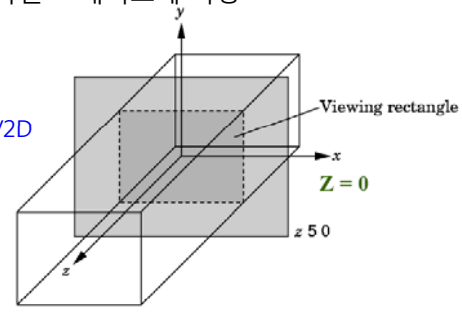
- OpenGL에서는 카메라가 물체의 공간(drawing coordinates)의 원점(origin)에 위치하며 z- 방향으로 향하고 있다.
- 관측공간을 지정하지 않는다면, 디폴트로 2x2x2 입방체의 viewing volume을 사용한다.



Viewing functions

- 직교 투영 (Orthographic parallel projection)
  - void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);
  - void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top); - 2차원 그래픽스에 사용

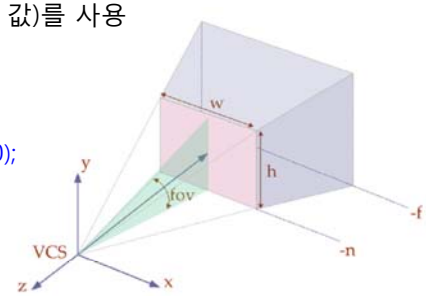
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 50.0, 0.0, 50.0); //2D
glMatrixMode(GL_MODELVIEW);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-10, 10, -10, 10, -10, 10);
glMatrixMode(GL_MODELVIEW);
```



Viewing functions

- 원근 투영 (Perspective projection)
  - void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);
  - void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar); - 상하좌우값을 설정하는 대신 y방향의 시선각도 (FOV)와 종횡비(가까운 쪽 클리핑 평면의 너비를 높이로 나눈 값)를 사용

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0, 1.0, 1.0, 10.0);
glMatrixMode(GL_MODELVIEW);
```



Viewport functions

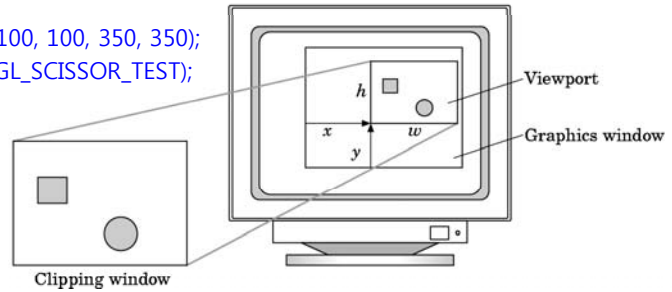
- 뷰포트 (Viewport)
  - 윈도우 내부에 설정한 공간. 그리기가 뷰포트 내부로 제한됨.
- void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)
  - 윈도우를 처음 생성할 때 전체 윈도우에 해당하는 픽셀 영역을 뷰포트로 설정; 이보다 작은 영역을 뷰포트로 설정할 때는 glViewport() 사용. 일반적으로 윈도우 전체를 뷰포트로 사용.
  - Reshape function이 있을 경우, glViewport()가 반드시 포함되어야 함.

```
glViewport(0, 0, glutGet(GLUT_WINDOW_WIDTH),
   glutGet(GLUT_WINDOW_HEIGHT));
```

Viewport functions

- void glScissor(GLint x, GLint y, GLsizei width, GLsizei height)
  - 윈도우의 직사각형 분할 및 그 분할 내에서의 그릴 때의 제한 사항 등을 정의
  - 일반적으로 glViewport()와 동일하게 정의함

```
glScissor(100, 100, 350, 350);
glEnable(GL_SCISSOR_TEST);
```

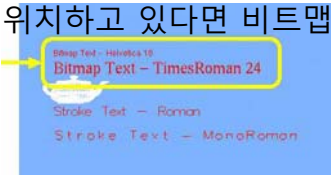


Text Functions

- GLUT에 정의된 문자 집합 제공. 문자열에는 획(stroke)과 래스터 (raster) 두 종류가 있다.
- Bitmap font (raster font) text
  - void glutBitmapCharacter(void *font, int character);
  - font 인자
    - 고정폭 폰트 - E.g. GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15
    - 10, 12, 18 포인트 비례 간격 폰트 – e.g. GLUT_BITMAP_TIMES_ROMAN_10
  - character인자는 ASCII
- Stroke font text
  - void glutStrokeCharacter(void *font, int character);
  - font 인자
    - GLUT_STROKE_ROMAN, GLUT_STROKE_MONO_ROMAN

Bitmap Text

- glutBitmapCharacter()는 비트맵 (bitmap = arrays of pixel data) 폰트를 사용하여 문자를 그린다.
- 비트맵 문자의 위치는 glRasterPos*를 사용하여 지정한다.
  - glRasterPos2f(x, y)
  - glRasterPos3f(x, y, z)
- 비트맵의 문자는 크기가 변하지 않는다. (unaffected by scaling or perspective)
- 화면에 일정한 곳에 위치하고 있다. 만약 래스터 위치가 뷰포트 밖에 위치하고 있다면 비트맵 문자는 그려지지 않는다.

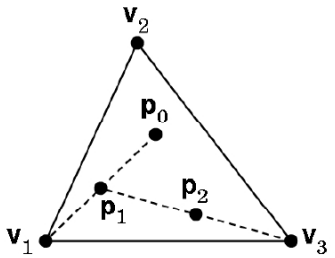


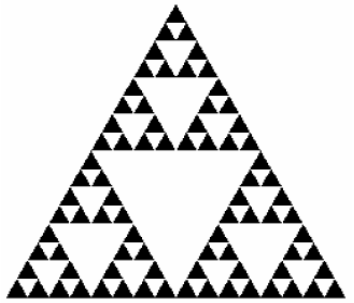
Stroke Text

- glutStrokeCharacter() 는 3차원 선으로 문자를 그린다.
- 따라서, GL의 변환 (즉, position, size, and orientation)에 의해 영향을 받는다.
- glutStrokeCharacter()는 다음 글자(character)를 그리기 위해 자체적으로 transformation을 한다.

Sierpinski Gasket

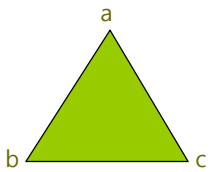
- A sample problem of drawing of the Sierpinski gasket
  1. Pick an initial point at random inside the triangle, P0
  2. Select one of the 3 vertices at random, v1
  3. Find the point halfway, P1
  4. Display this new point
  5. Replace the initial point with this new point
  6. Return to step 2



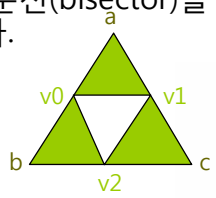


Sierpinski Gasket (2D)

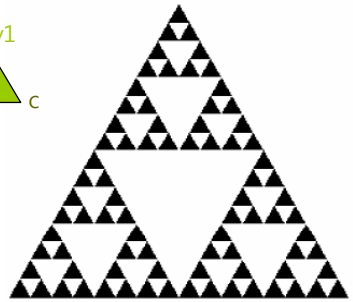
- 삼각형에서 시작한다.



- 삼각형 각 변의 이등분선(bisector)을 연결하고, 중앙의 작은 삼각형을 지운다.



- 반복 (Repeat)

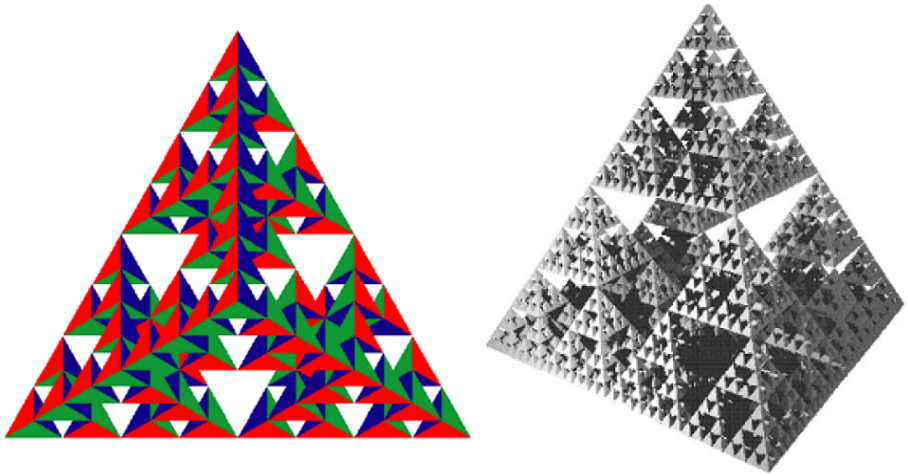


Sierpinski Gasket (2D)

```
/* recursive subdivision of triangle to form Sierpinski gasket
*/
#include <GL/glut.h>

/* initial triangle */
GLfloat v[3][2]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};
int n;

void triangle( GLfloat *a, GLfloat *b, GLfloat *c)
{
/* specify one triangle */
      glVertex2fv(a);
      glVertex2fv(b);
      glVertex2fv(c);
}

void divide_triangle(GLfloat *a, GLfloat *b, GLfloat *c, int m)
{
/* triangle subdivision using vertex numbers */
   GLfloat v0[2], v1[2], v2[2];
   int j;
   if(m>0)
   {
      for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
      for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
      for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
      divide_triangle(a, v0, v1, m-1);
      divide_triangle(c, v1, v2, m-1);
      divide_triangle(b, v2, v0, m-1);
   }
   else triangle(a,b,c); /* draw triangle at end of recursion */
}
```

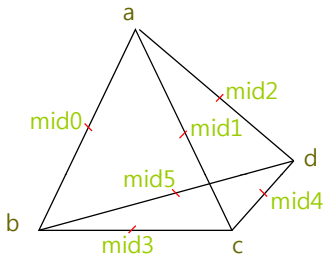
```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_TRIANGLES);
   divide_triangle(v[0], v[1], v[2], n);
   glEnd();
   glFlush();
}

void myinit()
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
   glMatrixMode(GL_MODELVIEW);
   glClearColor (1.0, 1.0, 1.0, 1.0);
   glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
   n=atoi(argv[1]); /* or set number of subdivision steps here
*/
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutCreateWindow("Sierpinski Gasket");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
}
```

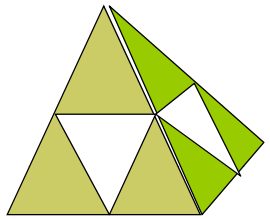
3D Gasket



3D Gasket

- 3D Gasket은 각각의 4면에서 subdivision을 한다.





- 사면체 (solid tetrahedron) 내부의 작은 사면체를 지운다.

3D Gasket

```
/* recursive subdivision of a tetrahedron to form 3D Sierpinski
gasket */
#include <stdlib.h>
#include <GL/glut.h>

/* initial tetrahedron */
GLfloat v[4][3]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
      {-0.816497, -0.471405, -0.333333},
      {0.816497, -0.471405, -0.333333}};
GLfloat colors[4][3] = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0},
                        {0.0, 0.0, 1.0}, {0.0, 0.0, 0.0}};
int n;
void triangle(GLfloat *va, GLfloat *vb, GLfloat *vc)
{
       glVertex3fv(va);
       glVertex3fv(vb);
       glVertex3fv(vc);
}

void tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d)
{
    glColor3fv(colors[0]);
    triangle(a, b, c);
    glColor3fv(colors[1]);
    triangle(a, c, d);
    glColor3fv(colors[2]);
    triangle(a, d, b);
    glColor3fv(colors[3]);
    triangle(b, d, c);
}
```

```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d,
int m)
{
    GLfloat mid[6][3];
    int j;
    if(m>0)
    {
        /* compute six midpoints */
        for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) mid[1][j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) mid[2][j]=(a[j]+d[j])/2;
        for(j=0; j<3; j++) mid[3][j]=(b[j]+c[j])/2;
        for(j=0; j<3; j++) mid[4][j]=(c[j]+d[j])/2;
        for(j=0; j<3; j++) mid[5][j]=(b[j]+d[j])/2;

        /* create 4 tetrahedrons by subdivision */
        divide_tetra(a, mid[0], mid[1], mid[2], m-1);
        divide_tetra(mid[0], b, mid[3], mid[5], m-1);
        divide_tetra(mid[1], mid[3], c, mid[4], m-1);
        divide_tetra(mid[2], mid[4], d, mid[5], m-1);
    }
    else(tetra(a,b,c,d)); /* draw tetrahedron at end of recursion */
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    divide_tetra(v[0], v[1], v[2], v[3], n);
    glEnd();
    glFlush();
}

void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w, 2.0 * (GLfloat) h / (GLfloat) w, -10.0, 10.0);
    else
        glOrtho(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

int main(int argc, char **argv)
{
    n=atoi(argv[1]); /* or enter number of subdivision steps here */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Z-buffer algorithm 사용

Using the Z-buffer

- 은면 제거 (Hidden surface removal)
  - 깊이 버퍼 (Z-buffer, depth buffer) 알고리즘
    - 픽셀 단위로 기하객체(geometry)의 z (깊이) 값이 가장 작은 평면의 값을 그린다. 픽셀당 깊이 정보 (depth value)를 저장하는 깊이 버퍼가 필요함
- Z-buffer 알고리즘
  - 깊이 버퍼를 초기화
    - glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)
  - 깊이 정보 테스트 활성화
    - glEnable(GL_DEPTH_TEST);
  - display callback에서 깊이 버퍼를 지움
    - glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

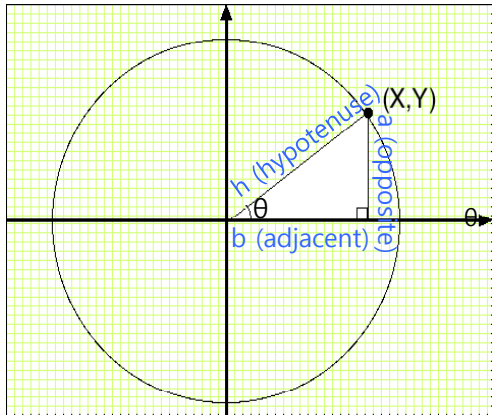
Geometric Primitives

Angles, Degrees, and Radians

- 일반적인 math library 함수는 radians을 사용함
- 360 degrees(°) = 1 full circle = 2 πradians
- 1 radian = 180.0/π degree ≈ 57.29578 degree
  or 1 degree = π/180.0 radian ≈ 0.01745329 radian

```
#ifndef M_PI
#define M_PI 3.141592654f
#endif
#define DegreesToRadians(degree) ((degree) * (M_PI / 180.0f))
#define RadiansToDegrees(radian) ((radian) * (180.0f / M_PI))
```

Trignometry



- $\sin\theta = a/h$
  $\cos\theta = b/h$
  $\tan\theta = a/b$
- $b = h*\cos\theta$
  $a = h*\sin\theta$
- $x^2 + y^2 = 1$
  $x = \cos\theta$
  $y = \sin\theta$
  $y/x = \sin\theta/\cos\theta = \tan\theta$
- x = distance * cosθ
  y = distance * sinθ

Trignometry

- Multiplicative inverse:
  $\csc\theta = 1/\sin\theta$
  $\sec\theta = 1/\cos\theta$
  $\cot\theta = 1/\tan\theta = \cos\theta/\sin\theta = x/y$
- Inverse:
  $\arcsin(x) = \sin^{-1}(x)$
  where $y=\arcsin(x)$ $x:[-1, 1] \rightarrow y:[-\pi/2, \pi/2]$
  $\arccos(x) = \cos^{-1}(x)$
  where $y=\arccos(x)$ $x:[-1, 1] \rightarrow y:[0, \pi]$
  $\arctan(x) = \tan^{-1}(x)$
  where $y=\arctan(x)$ $x:[-\infty, \infty] \rightarrow y:[-\pi/2, \pi/2]$

Trignometric Identity

- $\sin^2\theta + \cos^2\theta = 1$
  $1 + \tan^2\theta = \sec^2\theta$
  $1 + \cot^2\theta = \csc^2\theta$
- $\sin(\pi/2 - \theta) = \cos\theta$
  $\cos(\pi/2 - \theta) = \sin\theta$
  $\tan(\pi/2 - \theta) = \cot\theta$
- $\sin(x+y) = \sin x \cos y + \cos x \sin y$
  $\sin(x-y) = \sin x \cos y - \cos x \sin y$
  $\cos(x+y) = \cos x \cos y - \sin x \sin y$
  $\cos(x-y) = \cos x \cos y + \sin x \sin y$
- $\sin 2\theta = 2\sin\theta\cos\theta$
  $\cos 2\theta = \cos^2\theta - \sin^2\theta = 2\cos^2\theta - 1 = 1 - 2\sin^2\theta$

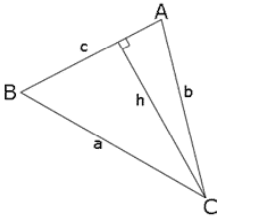
Law of Sines and Law of Cosines

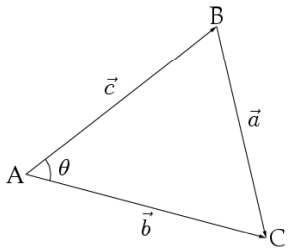
- Law of sines

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

- Law of cosines

$$a^2 = b^2 + c^2 - 2bc\cos A$$
$$b^2 = a^2 + c^2 - 2ac\cos B$$
$$c^2 = a^2 + b^2 - 2ab\cos C$$



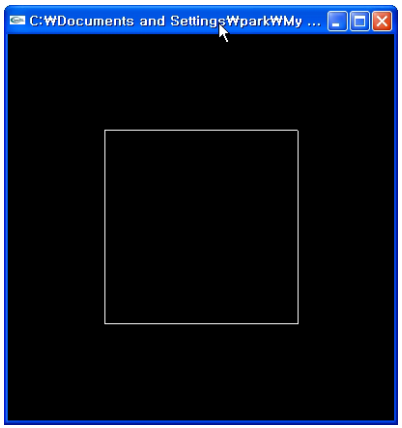


Geometric Primitives

- 가상의 물체를 표현하는데 있어 가장 기본이 되는 요소
- 실시간 그래픽스에서는 주로 단순한 형태의 표현방법인 선형 프리미티브 (Linear Primitives)를 사용
  - Point
  - Line, Line Segment, Ray
  - Sphere, Cylinder, Cone
  - Cube (Box)
  - Triangle
  - Polygon, ...
- OpenGL 에서의 다각형에 대한 요구 사항
  - 선분들이 서로 교차하면 안됨
  - 볼록 다각형만 사용해야 함
  - 꼭지점들이 한 평면 상에 존재해야 함

Square

- Draw a wireframe square
  - GL_LINE_LOOP
- Draw a solid square
  - GL_QUADS



Circle

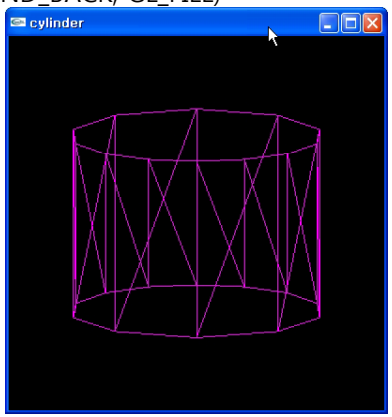
- Draw a wireframe circle
  - GL_LINE_LOOP
- Draw a solid circle
  - GL_TRIANGLE_FAN

```
void drawCircle(float radius, int step)
{
    float x, y, theta;
    theta = (float) (2*M_PI/step);
    glBegin(GL_LINE_LOOP);
    for (int i=0; i<step; i++) {
        x = radius * cos(theta * i);
        y = radius * sin(theta * i);
        glVertex3f(x, y, 0);
    }
    glEnd();
}
```

Cylinder

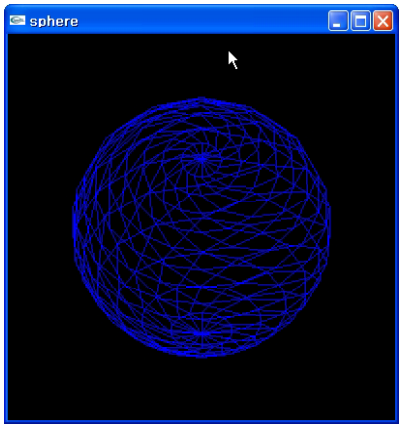
- Draw a wireframe cylinder
  - glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
- Draw a solid cylinder
  - glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)

```
void drawCylinder(float r, float h, int
step)
{
    float theta = (float) (2*M_PI/step);
    glBegin(GL_TRIANGLE_STRIP);
    for (int i=0; i<=step; i++) {
        float x = r * cos(theta * i);
        float z = r * sin(theta * i);
        glVertex3f(x, -h/2, z);
        glVertex3f(x, h/2, z);
    }
    glEnd();
}
```



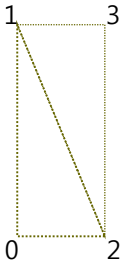
Sphere

- Draw a wireframe sphere
  - GL_LINE_STRIP
- Draw a solid sphere
  - GL_TRIANGLE_STRIP



Sphere

```
void drawSphere(float radius, int stacks, int slices)
{
    float lonstep = M_PI/stacks; float latstep = M_PI/slices;
    for (lon = 0.0; lon<=2*M_PI; lon+= (lonstep)) {
        glBegin(GL_TRIANGLE_STRIP);
        for (lat=0.0; lat<=M_PI+latstep; lat += (latstep)) {
            x = cosf(lon)*sinf(lat)*radius;
            y = sinf(lon)*sinf(lat)*radius;
            z = cosf(lat)*radius;
            glNormal3f(x, y, z);
            glVertex3f(x, y, z);
            x = cosf(lon+lonstep)*sinf(lat)*radius;
            y = sinf(lon+lonstep)*sinf(lat)*radius;
            z = cosf(lat)*radius;
            glNormal3f(x,y,z);
            glVertex3f(x, y, z);
        }
        glEnd();
    }
}
```



경도(lon) 위도(lat)

x = cosφ *cosθ
y = sinθ
z = sinφ * cosθ
where 0 ≤ φ ≤ 2π, -π/2 ≤θ≤ π/2

Cube

- Draw a wireframe cube
  - GL_LINE_LOOP
- Draw a solid cube
  - GL_QUADS