

Geometric Objects and Transformation

321190
2011년 봄학기
4/19/2011
박경신

OpenGL Transformation

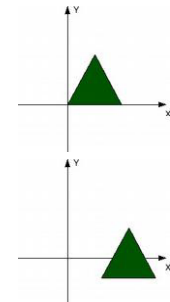
- OpenGL은 기본적인 변환을 수행하는 함수를 제공한다.
- *Translation*: 이동변환은 3차원 이동 변위벡터 (dx, dy, dz)를 넣는다.
- *Rotation*: 회전변환은 axis(회전축)와 angle(각도)을 넣는다.
- *Scaling*: 크기변환은 scaling factor (크기변환 값)를 넣는다.
- 이 각각의 함수는 4x4 변환 행렬을 생성하여 객체에 적용이 된다.

OpenGL Transformation

- `glTranslate*(dx, dy, dz)`
 - 이동변환 인자 dx, dy, dz는 실수
 - 2차원 이동 : dz = 0.0
 - `glTranslatef(25.0, -10.0, 0.0);`
- `glRotate* (theta, vx, vy, vz)`
 - 회전변환 인자 theta는 회전량 각도 (rotation angle in degrees)는 0~360 사이의 값을 가짐
 - 벡터 (vx, vy, vz)는 회전축 (rotation axis)
 - `glRotatef(90, 0, 0, 1);`
- `glScale*(sx, sy, sz)`
 - 크기변환 인자 sx, sy, sz는 실수
 - 반사 (reflection)는 인자값에 음수를 적용
 - `glScalef(2.0, -3.0, 1.0);`
- Suffix code (*)는 f (float) 또는 d (double)

Translation in OpenGL

- `glTranslatef(x, y, z)`
 - 객체를 (x, y, z)만큼 이동
 - 2차원 `glTranslate` 함수는 없다.
- Example:
`glTranslatef(0.5, -0.2, 0.0);`
`glBegin(GL_TRIANGLES);`
`glVertex2f(0.0, 0.0);`
`glVertex2f(0.5, 0.0);`
`glVertex2f(0.25, 0.5);`
`glEnd();`



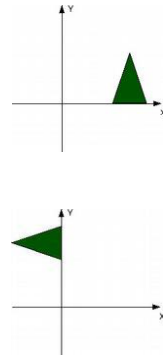
Rotation in OpenGL

□ glRotatef(angle, x, y, z)

- 임의의 축 axis (x,y,z)에 대해 각도 angle 만큼 회전
- 2차원 회전은 z-축 (0, 0, 1)로 사용한다.

□ Example:

```
glRotatef(90.0, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
glVertex2f(0.5, 0.0);
glVertex2f(0.8, 0.0);
glVertex2f(0.65, 0.5);
glEnd();
```



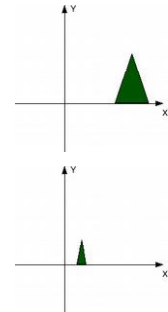
Scale in OpenGL

□ glScalef(x, y, z)

- x-축으로 x만큼, y-축으로 y만큼, z-축으로 z만큼 크기를 변환
- 이 때, scale factor > 1이면 커지고, 0 < scale factor ≤ 1이면 작아지고, scale factor < 0면 반사(reflection)된다.
- 2차원 크기변환은 z에 1을 넣는다.

□ Example:

```
glScalef(0.25, 0.5, 1.0);
glBegin(GL_TRIANGLES);
glVertex2f(0.5, 0.0);
glVertex2f(0.8, 0.0);
glVertex2f(0.65, 0.5);
glEnd();
```

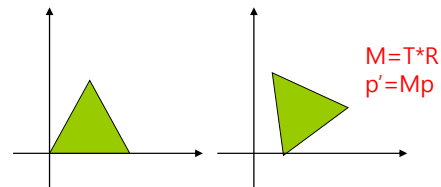


Transformation Order

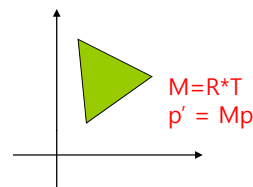
□ OpenGL에서 모델링 변환 행렬들은 객체에 설정된 반대 순서로 적용된다.

- 즉, 마지막 변환 (즉, 기하 함수 호출 바로 전에 쓰인 것)이 정점 데이터에 먼저 적용된다.

```
glTranslatef(0.5, 0, 0);
glRotatef(45, 0, 0, 1);
drawTriangle();
```



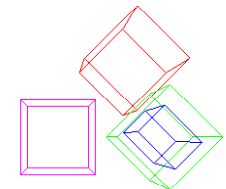
```
glRotatef(45, 0, 0, 1);
glTranslatef(0.5, 0, 0);
drawTriangle();
```



Transformation Order

```
glColor3f(1.0, 0.0, 1.0);
glutWireCube(1);
```

```
glPushMatrix();
glRotatef(45.0, 0.0, 0.0, 1.0);
glTranslatef(1.5, 0.0, 0.0);
glColor3f(1.0, 0.0, 0.0);
glutWireCube(1);
glPopMatrix();
```



```
glPushMatrix();
glTranslatef(1.5, 0.0, 0.0);
glRotatef(45.0, 0.0, 0.0, 1.0);
glColor3f(0.0, 1.0, 0.0);
glutWireCube(1);
glPopMatrix();
```

Matrix Operations

■ glMatrixMode(GLenum mode)

- glMatrixMode는 행렬 모드를 설정한다. 현재의 행렬이 모델뷰 (model view), 투영 (projection), 텍스처 (texture) 행렬 중 어떤 것인지를 나타내는 함수이다.

■ glMatrixMode(GL_PROJECTION)

- 연속되는 행렬 연산을 투영 행렬 (*projection matrix*) 스택에 적용한다. 투영 행렬은 3D 공간에 카메라 설정을 수학적으로 표현한 행렬이다.

■ glMatrixMode(GL_MODEL_VIEW)

- 연속되는 행렬 연산을 기하 변환 행렬 (*geometric transformation matrix*) 스택에 적용한다. 3D 공간에 물체의 배치를 수학적으로 표현한 행렬이다.
- 현재 modelview 모드에 있으면, 모델링 변환 함수 (즉, 이동, 회전, 크기변환 함수)의 행렬을 기하변환 행렬 스택에 적용한다.

Matrix Operations

■ glLoadIdentity()

- 현재의 행렬을 4x4 단위행렬 (identity matrix)로 설정한다. 즉, 현재의 행렬을 초기화 한다.

■ glLoadMatrix{d/f}(const GLdouble/GLfloat *M)

- 현재 행렬에 임의의 변환행렬 16개의 값 M을 대입한다.
- 이 때, M의 element는 열-중심 (column-major) 순서로 지정해야 한다.

$$M = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Matrix Operations

```
glMatrixMode(GL_MODELVIEW);  
GLfloat M[16];  
GLint k;  
for (k=0; k<16; k++)  
    elements[k] = float(k);  
glLoadMatrixf(M);
```

$$M = \begin{pmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{pmatrix}$$

Matrix Operations

■ glMultMatrix{d/f}(const GLdouble/GLfloat *M)

- 현재의 행렬에 M을 곱한다.
- 이 때, M의 16개의 값을 가지고 있고, 각 element는 열-중심 (column-major) 순서로 지정해야 한다.
- 현재의 행렬은 glMultMatrix에서 지정한 행렬에 의해 postmultiply 된 후 값이 바뀐다.

```
// 물체를 공간상에 그리기위해 행렬모드를 GL_MODELVIEW로 설정  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity(); // 행렬을 초기화  
glMultMatrixf(M2); // M2행렬을 곱함  
glMultMatrixf(M1); // M1행렬을 곱함
```

Matrix Operations

□ glGetFloatv(GL_MODELVIEW_MATRIX, M)

- 현재의 변환 행렬을 M으로 돌려준다.

```
GLfloat M[16];  
glGetFloatv(GL_MODELVIEW, M);
```

□ glPushMatrix() & glPopMatrix()

- 행렬 스택에 glPushMatrix()는 현재 상태를 저장하고
- glPopMatrix()는 스택의 맨 위에 저장된 상태로 복원시킨다.

Matrix Operations

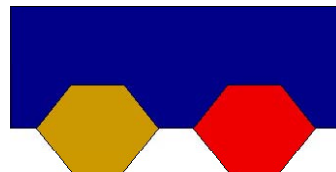
- $C \leftarrow I$ `glLoadIdentity();`
- $C \leftarrow C T$ `glTranslatef(dx, dy, dz);`
- $C \leftarrow C S$ `glScalef(sx, sy, sz);`
- $C \leftarrow C R$ `glRotatef(angle, ax, ay, az);`
- $C \leftarrow M$ `glLoadMatrixf(ptr_to_matrix);`

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();           // C = I  
glTranslatef(2.0, 1.0, 0.0); // C = T(p)  
glRotatef(60, 1.0, 0.0, 0.0); // C = C R(q) = T(p) R(q)  
glTranslatef(-2.0, 1.0, 0.0); // C = C T(-p) = T(p) R(q) T(-p)  
drawObject();
```

Hierarchical Transformations

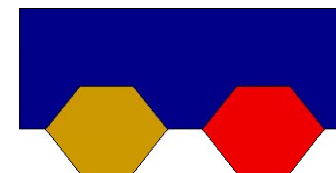
- 계층적 변환 (hierarchical transformation)은 한 변환을 다른 변환에 소속시키는 것으로 생각하면 된다.
- 계층적 변환이란 한 객체의 변환을 다른 객체들에 상대적인 변환으로 사용된다.
- 2개의 자동차 바퀴(wheel)가 자동차 차체(body)에 상대적인 계층적 변환의 예를 보자면:

- Apply body transformation
- Draw body
- Save state
- Apply front wheel transformation
- Draw wheel
- Restore saved state
- Apply rear wheel transformation
- Draw wheel



Hierarchical Transformations

- 또한, 이 자동차가 움직이게 되면, 자동차의 차체에서 상대적인 위치에 있는 바퀴 두 개도 역시 몸체와 같이 움직이게 됨을 알 수 있다.
- 이 때, 두 개의 바퀴를 자동차의 몸체의 변환에 같이 영향을 받도록 만들어 하며, 바퀴가 각자 따로 변환하지 않도록 한다.



Example: Car

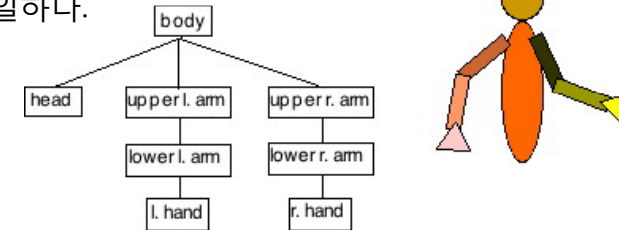
```
glLoadIdentity();
glColor3f(0.0, 0.0, 1.0);
glTranslatef(carpos[0], carpos[1], carpos[2]);
drawBody();

glColor3f(1.0, 1.0, 0.0);
glPushMatrix();
glTranslatef(-0.2, 0.0, 0.0);
glRotatef(angle, 0, 0, 1);
drawWheel();
glPopMatrix();

glColor3f(1.0, 0.0, 0.0);
glPushMatrix();
glTranslatef(0.2, 0.0, 0.0);
glRotatef(angle, 0, 0, 1);
drawWheel();
glPopMatrix();
```

Transformation Hierarchy

- 계층적 변환 (hierarchical transformations)은 종종 변환의 트리 (tree) 구조로 표현한다.
- 3차원 캐릭터를 디자인하기 위해 강체 부분 (rigid body parts)으로 만들어진 계층적 변환 구조를 사용한다.
- 그리고, 보다 유연한 3차원 캐릭터 디자인을 위해서는 다수의 계층적 변환을 적절히 섞어 사용해야 한다.
- 이런 계층은 장면그래프 (scene graph)의 기초와 동일하다.



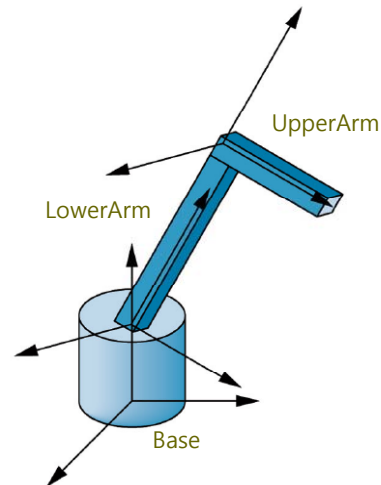
Example: Robot

```
Display()
{
    glPushMatrix();
    glRotatef (theta, 0.0, 1.0, 0.0);
    Base();

    glTranslatef (0.0, h1, 0.0);
    glRotatef(phi, 0.0, 0.0, 1.0);
    DrawLowerArm ();

    glTranslatef (0.0, h2, 0.0);
    glRotatef (psi, 0.0, 0.0, 1.0);
    DrawUpperArm();

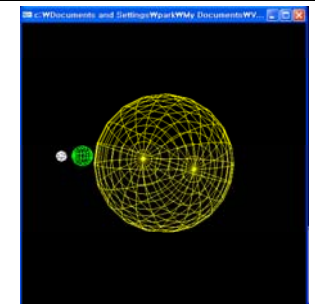
    glPopMatrix();
}
```



Example: Solar

```
// global variables
float g_ElapsedTime;
double g_CurrentTime, g_PreviousTime;
float g_SunRadius = 5.0f;
float g_EarthRadius = 1.0f;
float g_MoonRadius = 0.5f;
float g_EarthDistanceFromSun = -12.0f;
float g_MoonDistanceFromEarth = -2.0f;
```

```
void update()
{
    g_CurrentTime = timeGetTime();
    g_ElapsedTime = (float)((g_CurrentTime - g_PreviousTime) * 0.001);
    g_PreviousTime = g_CurrentTime;
    glutPostRedisplay();
}
```



Example: Solar

```
void display (void)
{
    ...
    static float SunSpin    = 0.0f;
    static float EarthSpin  = 0.0f;
    static float EarthOrbit = 0.0f;
    static float MoonSpin   = 0.0f;
    static float MoonOrbit  = 0.0f;
    if( g_OrbitOn == true )
    {
        SunSpin -= g_Speed * (g_ElapsedTime * 10.0f);
        EarthSpin -= g_Speed * (g_ElapsedTime * 100.0f);
        EarthOrbit -= g_Speed * (g_ElapsedTime * 20.0f);
        MoonSpin -= g_Speed * (g_ElapsedTime * 50.0f);
        MoonOrbit -= g_Speed * (g_ElapsedTime * 200.0f);
    }
}
```

Example: Solar

```
// The Sun (spins by rotating it about y-axis)
glPushMatrix();
    glRotatef( SunSpin, 0.0f, 1.0f, 0.0f ); // spin on its own axis.
    glColor3f( 1.0f, 1.0f, 0.0f );
    glutWireSphere( g_SunRadius, 20, 20 );
glPopMatrix();

// The Earth spins on its own axis and orbit the Sun.
glPushMatrix();
    glRotatef( EarthOrbit, 0.0f, 1.0f, 0.0f );
    glTranslatef( 0.0f, 0.0f, g_EarthDistanceFromSun );
    glRotatef( EarthSpin, 0.0f, 1.0f, 0.0f );
    glColor3f( 0.0f, 1.0f, 0.0f );
    glutWireSphere( g_EarthRadius, 10, 10 );
glPopMatrix();
```

Example: Solar

```
// The Moon spins on its own axis and orbit the Earth.
glPushMatrix();
    glRotatef( EarthOrbit, 0.0f, 1.0f, 0.0f );
    glTranslatef( 0.0f, 0.0f, g_EarthDistanceFromSun );

    glRotatef( MoonOrbit, 0.0f, 1.0f, 0.0f );
    glTranslatef( 0.0f, 0.0f, g_MoonDistanceFromEarth );
    glRotatef( MoonSpin, 0.0f, 1.0f, 0.0f );

    glColor3f( 1.0f, 1.0f, 1.0f );
    glutWireSphere( g_MoonRadius, 8, 8 );
glPopMatrix();

glutSwapBuffers();
}
```

Example: Solar Using Matrix & Vector Class

```
#include "matrix4x4.h"
#include "vector3.h"
...
void display (void)
{
    ...

    // The Sun (spins by rotating it about y-axis)
    matrix4x4 mSunSpinRotation;
    matrix4x4 mSunMatrix;
    mSunSpinRotation.rotate( SunSpin, 'y');
    mSunMatrix = mSunSpinRotation; // spin it on its axis.

    glPushMatrix();
        glMultMatrixf( mSunMatrix.m );
        glColor4f( 1.0f, 1.0f, 0.0f, 1.0f );
        glutWireSphere( g_SunRadius, 20, 20 );
    glPopMatrix();
}
```

Example: Solar Using Matrix & Vector Class

```
// The Earth spins on its own axis and orbit the Sun.
matrix4x4 mEarthTranslationToOrbit;
matrix4x4 mEarthSpinRotation;
matrix4x4 mEarthOrbitRotation;
matrix4x4 mEarthMatrix;
mEarthSpinRotation.rotate( EarthSpin, 'y' );
mEarthTranslationToOrbit.translate(vector3(0.0f, 0.0f,
g_EarthDistanceFromSun));
mEarthOrbitRotation.rotate( EarthOrbit, 'y' );
mEarthMatrix = mEarthOrbitRotation *
                mEarthTranslationToOrbit *
                mEarthSpinRotation;

glPushMatrix();
    glMultMatrixf( mEarthMatrix.m );
    glColor4f( 0.0f, 1.0f, 0.0f, 1.0f );
    glutWireSphere( g_EarthRadius, 10, 10 );
glPopMatrix();
```

Example: Solar Using Matrix & Vector Class

```
// The Moon spins on its own axis and orbit the Earth
matrix4x4 mMoonTranslationToOrbit;
matrix4x4 mMoonSpinRotation;
matrix4x4 mMoonOrbitRotation;
matrix4x4 mMoonMatrix;
mMoonSpinRotation.rotate( MoonSpin, 'y' );
mMoonTranslationToOrbit.translate( vector3(0.0f, 0.0f, g_MoonDistanceFromEarth) );
mMoonOrbitRotation.rotate( MoonOrbit, 'y' );
mMoonMatrix = mEarthOrbitRotation *
                mEarthTranslationToOrbit *
                mMoonOrbitRotation *
                mMoonTranslationToOrbit *
                mMoonSpinRotation;

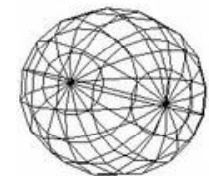
glPushMatrix();
    glMultMatrixf( mMoonMatrix.m );
    glColor4f( 1.0f, 1.0f, 1.0f, 1.0f );
    glutWireSphere( g_MoonRadius, 8, 8 );
glPopMatrix();
glutSwapBuffers();
}
```

3D Geometry Functions

- GLUT shapes
- GLU quadrics
- Modeling a cube
- 3D Model Loading

GLUT Shapes

- GLUT는 다양한 기본 도형 드로잉 함수를 제공한다. 예를 들어, platonic solids, simple curves, teapots 등등
- GLUT 도형 드로잉함수는 solid나 wireframe으로 그릴 수 있다.
 - glutSolidSphere(1.5, 16, 8)
 - glutWireDodecahedron()
- 예제
 - glutShapes.cpp



GLUT Shapes

- `glutSolidCube(size)`
- `glutSolidSphere(radius, slices, stacks)`
- `glutSolidCone(baseRadius, height, slices, stacks)`
- `glutSolidTorus(innerRadius, outerRadius, sides, rings)`
- `glutSolidOctahedron()`
- `glutSolidDodecahedron()`
- `glutSolidIcosahedron()`
- `glutSolidTeapot(size)`

GLU Quadrics

- Quadrics이란 예를 들어 $x^2 + y^2 + z^2 = r^2$ 와 같은 다양한 곡선이나 표면을 만들어 내는 함수이다.
- 기본적인 GLU quadrics 는 spheres, cylinders, cone, disk를 포함하고 있다.
- GLU quadrics를 그리려면 quadric object을 생성해서 GLU 함수에 넘겨줘야 한다. 그리고, GLU에서는 quadrics를 points, lines, 또는 polygons 등 어떻게 그릴지 제어하는 함수도 제공하고 있다.

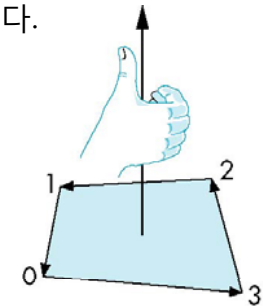
```
quadric = gluNewQuadric();  
gluQuadricDrawStyle(quadric, GLU_LINE);  
gluSphere(quadric, 2.5, 32, 24);
```
- 예제
 - `gluQuadrics.cpp`

GLUT Quadrics

- `gluSphere(quadric, radius, slices, stacks)`
- `gluCylinder(quadric, baseRadius, topRadius, height, slices, stacks)`
- Cone using `gluCylinder(quadric, 0, topRadius, height, slices, stacks)`
- `gluDisk(quadric, innerRadius, outerRadius, slides, rings)`
- `glutPartialDisk(quadric, innerRadius, outerRadius, slides, rings, startAngle, sweepAngle)`

Modeling a Cube

- OpenGL에서 정점의 winding 순서 $\{v_0, v_3, v_2, v_1\}$ 과 $\{v_1, v_0, v_3, v_2\}$ 은 같은 다각형을 만들어낸다. 그러나, 정점의 winding 순서 $\{v_1, v_2, v_3, v_0\}$ 은 다르다.
- OpenGL에서는 오른손 좌표계를 사용하므로, counter-clockwise encirclement로 정점을 정의했을 때 바깥쪽을 향하는 법선벡터 (normal)을 만들어낸다.

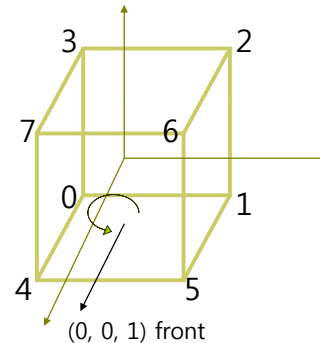


Modeling a Cube

- Vertex list와 Index list를 사용하여 cube를 그린다.

```
GLfloat vertex[][3] = {
    {-1.0,-1.0,-1.0}, { 1.0,-1.0,-1.0},
    { 1.0, 1.0,-1.0}, {-1.0, 1.0,-1.0},
    {-1.0,-1.0, 1.0}, { 1.0,-1.0, 1.0},
    { 1.0, 1.0, 1.0}, {-1.0, 1.0, 1.0}
};

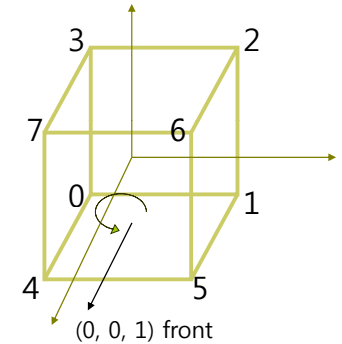
GLfloat normal[][3] = {
    { 1.0, 0.0, 0.0}, // right
    { 0.0, 1.0, 0.0}, // top
    { 0.0, 0.0, 1.0}, // front
    {-1.0, 0.0, 0.0}, // left
    { 0.0, -1.0, 0.0}, // bottom
    { 0.0, 0.0, -1.0}, // back
};
```



Modeling a Cube

```
void polygon(int n, int b, int c, int d)
{
    glBegin(GL_POLYGON);
    glNormalfv(normal[n]);
    glVertex3fv(vertex[a]);
    glVertex3fv(vertex[b]);
    glVertex3fv(vertex[c]);
    glVertex3fv(vertex[d]);
    glEnd();
}

void cube( )
{
    polygon(0, 5, 1, 2, 6); // right
    polygon(1, 6, 2, 3, 7); // top
    polygon(2, 6, 7, 4, 5); // front
    polygon(3, 4, 7, 3, 0); // left
    polygon(4, 4, 0, 1, 5); // bottom
    polygon(5, 0, 3, 2, 1); // back
}
```



Model Files

- 3차원 모델 종류
 - Wavefront (.obj)
 - Inventor (.iv)
 - VRML / X3D
 - 3D Studio (.3ds)
 - OpenFlight (.flt)
 - ...
- 3차원 객체 모델은 아래와 같은 정보를 포함하고 있다.
 - Geometry data – vertex positions, faces
 - Colors/material properties
 - Textures
 - Transformations

Wavefront OBJ Files

- OBJ file은 일반 텍스트 파일로, 정점 (vertices), 다각형 표면 (polygon faces), 재질 (material) 등 그 외 다수의 정보를 포함하고 있다.
- 각 line은 정점, 법선벡터, 텍스처 등 어떤 정보를 가진 line인지 알려주는 토큰 (token)으로 시작한다.
 - v x y z
 - Vertex position
 - vn x y z
 - Vertex normal
 - vt u v
 - texture coordinate
 - f v1 v2 v3 ..
 - Face (list of vertex numbers)
 - Mtllib *file.mtl*
 - File containing material descriptions
 - Usemtl *name*
 - Current material to apply to geometry