GLUT Modeling, Transformation, and Bezier Curve

동아대학교 컴퓨터AI공학부

박영진

GLUT Modeling

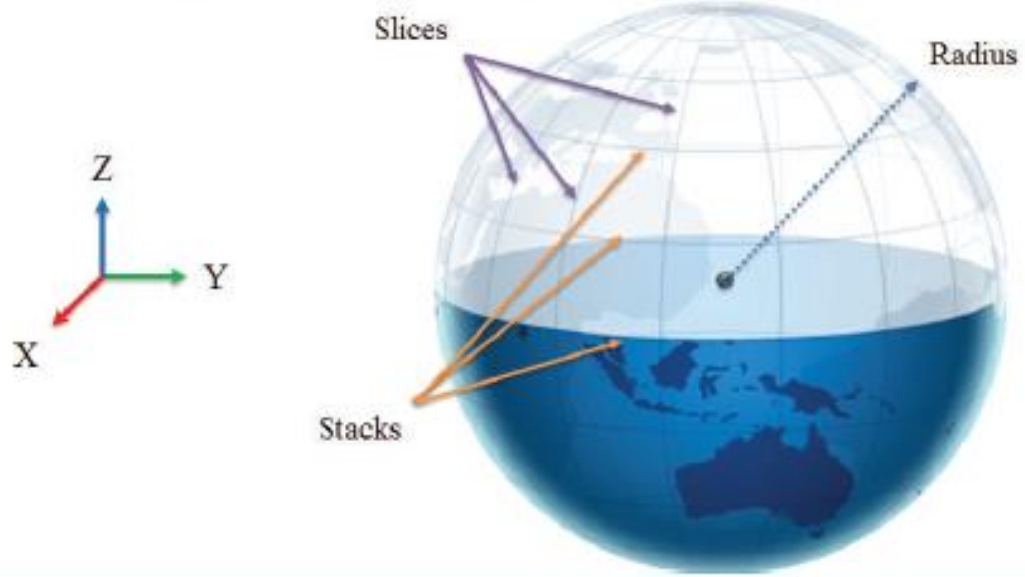
- **GLUT Library를 이용한 3D Primitive 시각화**
- **그리는 방식**
  - Wireframe Models
  - Solid Models
- **기본 도형**
  - Cube : glutWireCube(), glutSolidCube()
  - Cone : glutWireCone(), glutSolidCone()
  - Sphere, Torus, Tetrahedron, Octahedron, ...
  - Utah Teapot

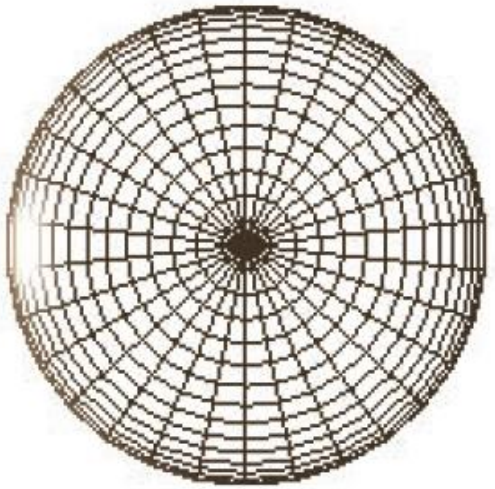
Sphere

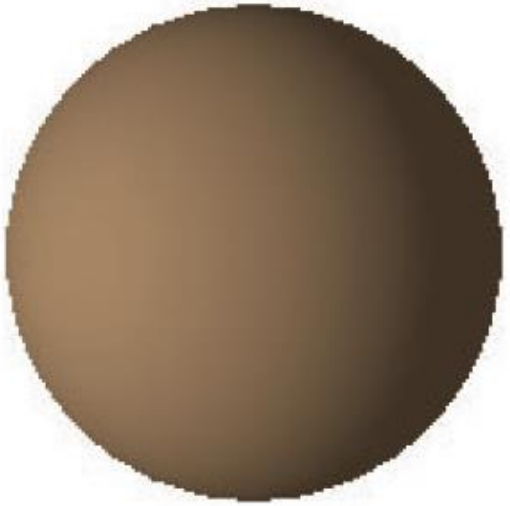
```
void glutWireSphere ( GLdouble radius, GLint slices, GLint stacks );
void glutSolidSphere ( GLdouble radius, GLint slices, GLint stacks );
```

Parameters Help

radius // Sphere의 반지름(Radius)
slices // Z축 주변의 분할(Divisions)의 개수
stacks // Z축을 따르는 분할(Divisions)의 개수



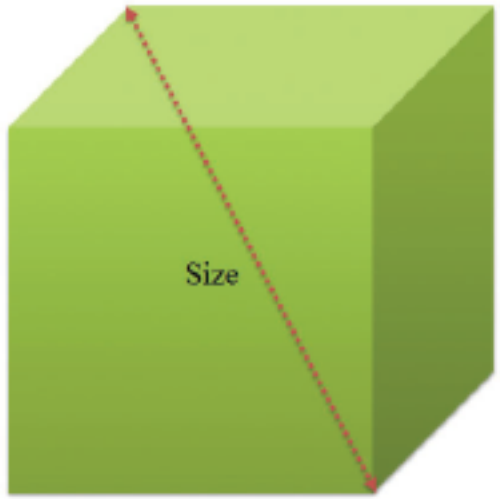


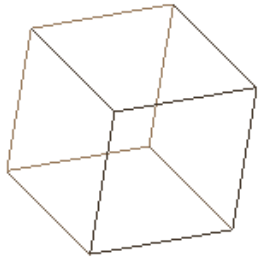


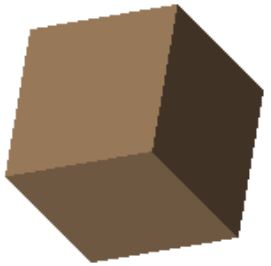
Cube

void glutWireCube (Gldouble *size*);
void glutSolidCube (Gldouble *size*);

Parameters Help	*size* // 동일한 넓이(Width), 높이(Height) 및 깊이(Depth) 값을 가지는 Cube의 양쪽 꼭짓점 사이의 직경(Diameter, 지름)





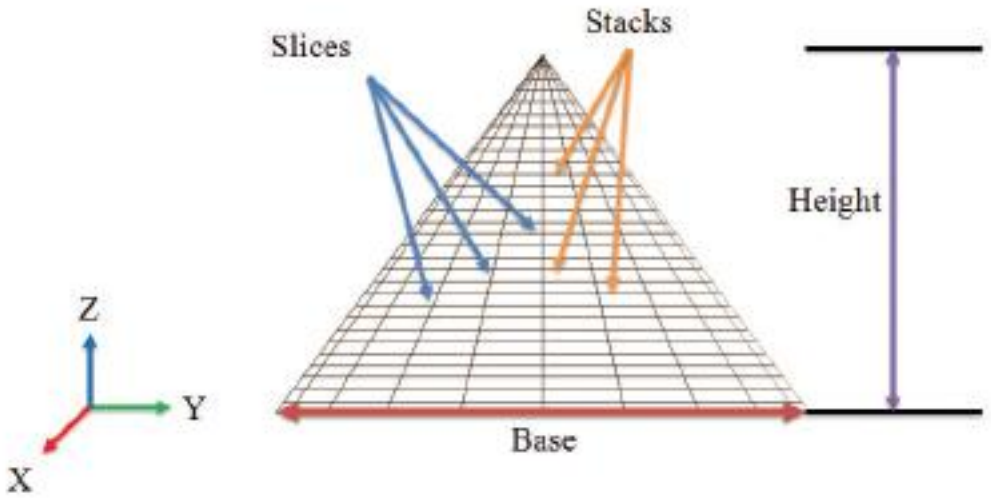


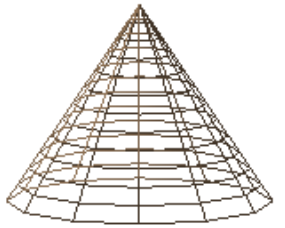
Cone

```
void glutWireCone ( GLdouble base, GLdouble height, GLint slices, GLint stacks );
void glutSolidCone ( GLdouble base, GLdouble height, GLint slices, GLint stacks );
```

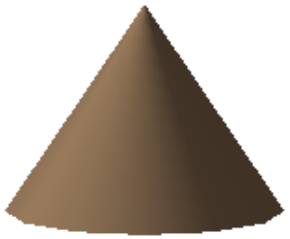
Parameters Help

base // XY 평면을 기준으로 하는 Cone의 직경(Diameter, 지름)
height // Z축 방향으로의 Cone의 높이(Height)
slices // Z축 주변의 분할(Division)의 개수(경도, Longitude)
stacks // Z축을 따르는 분할(Division)의 개수(위도, Latitude)





glutWireCone(1.0, 1.5, 12, 12);



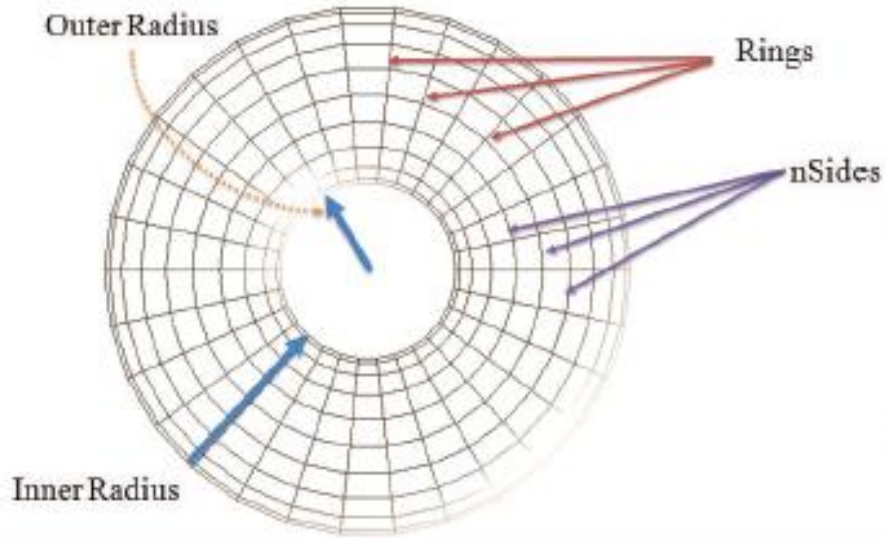
glutSolidCone(1.0, 1.5, 12, 12);

Torus

void glutWireTorus (GldoubleinnerRadius, Gldouble outerRadius, Glint nsides, Glint rings);
void glutSolidTorus (GldoubleinnerRadius, Gldouble outerRadius, Glint nsides, Glint rings);

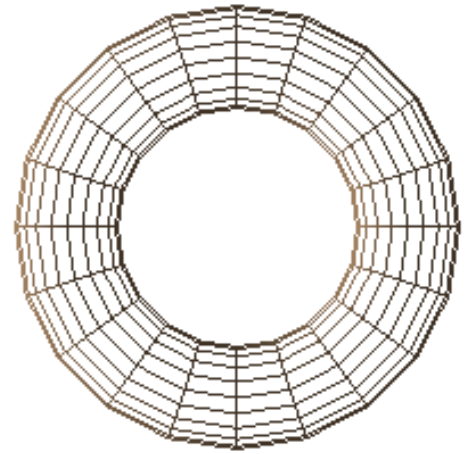
Parameters Help

innerRadius // Torus의 안쪽 방향의 직경(Diameter, 지름)
outerRadius // Torus의 바깥쪽 방향의 직경(Diameter, 지름)
nsides // 각 원형 단면을 위한 면(Face)들의 개수
rings // Torus를 위한 원형 분할(Division)의 개수

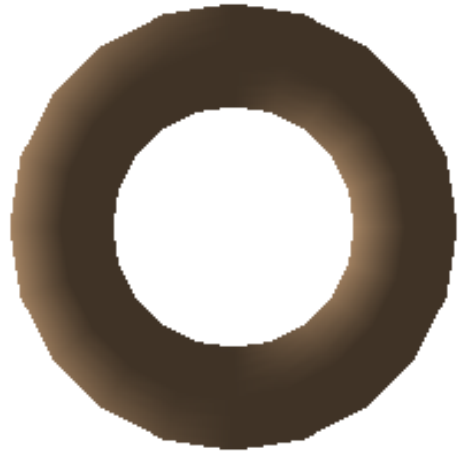


Inner Radius : Torus의 바깥쪽 경계 부분에서 원점 방향으로 Torus의 안쪽 원까지의 직경(지름)

Outer Radius : Torus의 안쪽 원점에서 Torus의 안쪽 원까지의 직경(지름)



glutWireTorus(0.3, 1.5, 20, 20);



glutSolidTorus(0.3, 1.5, 20, 20);

Utah Teapot

```
void glutSolidTeapot ( GLdouble size );
void glutWireTeapot ( GLdouble size );
```

Parameters	*size* // Scale factor 즉, 어떤 양(Quantity)이나 곱셈(Multiply) 혹은
Help	// 축척(Scale)을 의미하는 숫자이다.

glutWireTeapot(1.0);

glutSolidTeapot(1.0);

Geometric Transformation

- 이동 (Translation)
- 회전 (Rotation)
- 크기 조절 (Scale)
- 복합 변환

- 이동변환에 대한 이론 강의는 영상 업로드 예정

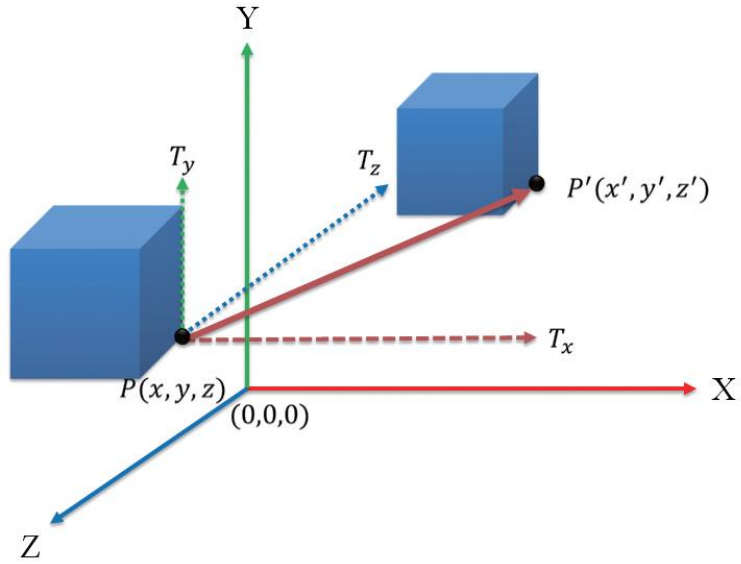
이동 Translation

- 2D와 3D 공간에서 강체(Rigid Body)의 이동(Translation)

$$P' = T + P \qquad P' = T \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} T_x + x \\ T_y + y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \\ 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} T_x + x \\ T_y + y \\ T_z + z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



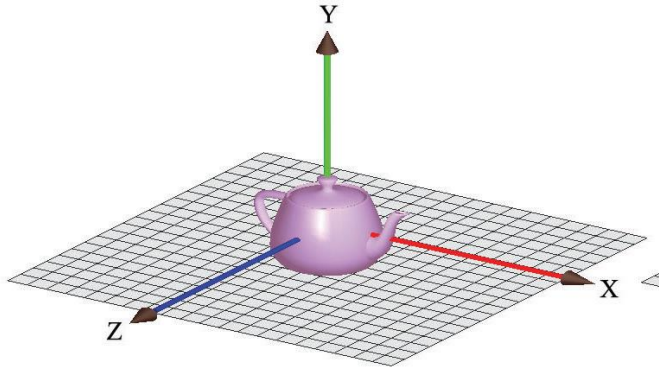
void glTranslatef (GLfloat *x*, GLfloat *y*, GLfloat *z*);	
Parameters Help	*x* // X축으로 이동하기 위한 값 *y* // Y축으로 이동하기 위한 값 *z* // Z축으로 이동하기 위한 값

glTranslatef 함수를 사용한 객체의 이동

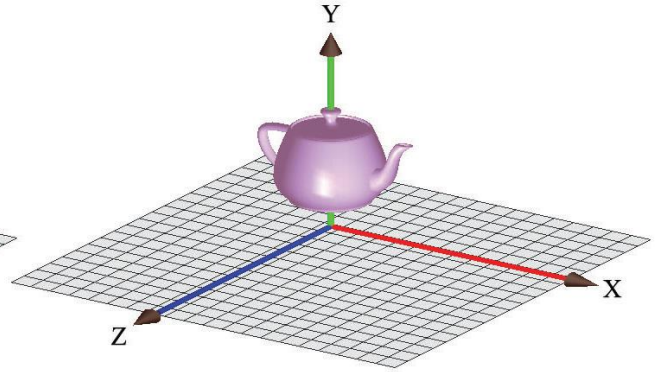
```
...
glTranslatef(2.0, 3.0, 1.0);
glutSolidTeapot(1.0);
...
```

이동 Translation

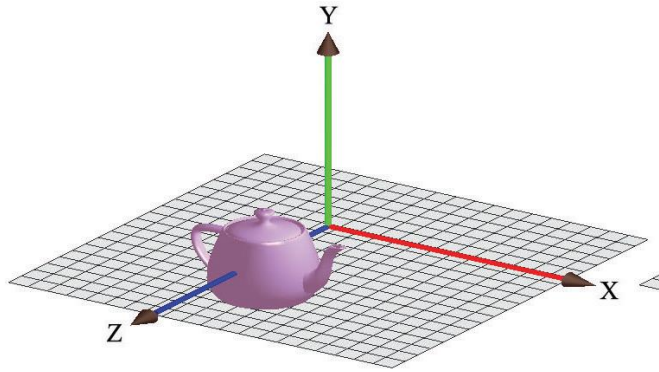
- 3D 공간에서 강체의 이동(Translation) 구현 결과



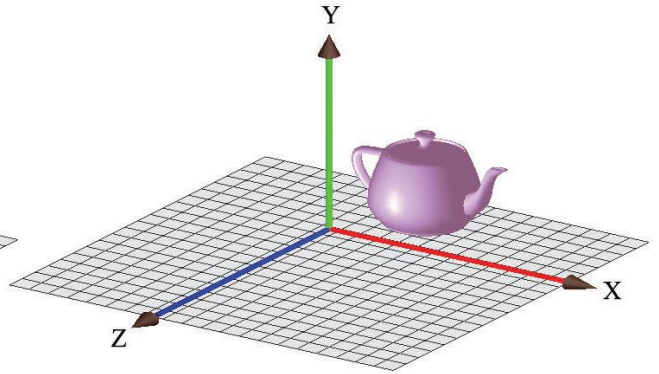
(A) glTranslatef(0.0, 0.0, 0.0);



(B) glTranslatef(0.0, 1.0, 0.0);



(C) glTranslatef(0.0, 0.0, 1.0);



(D) glTranslatef(1.0, 1.0, 0.0);

회전 Rotation

- 2D 공간에서 강체의 회전

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

- 3D 공간에서 강체의 회전

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

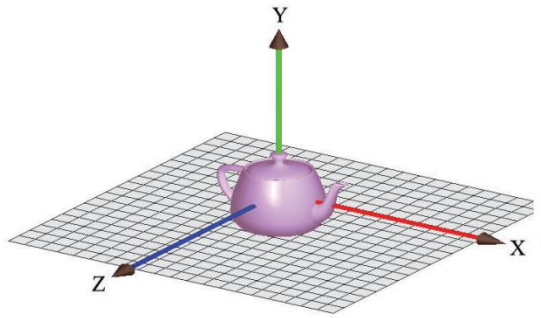
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

void glRotatef (GLfloat *angle*, GLfloat *x*, GLfloat *y*, GLfloat *z*);

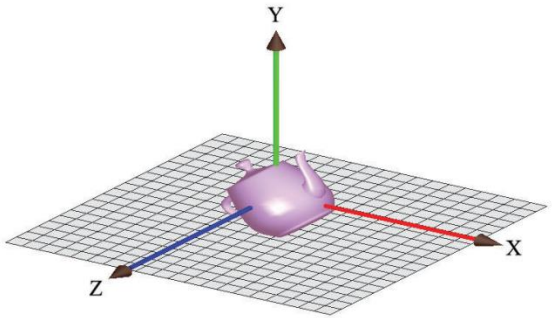
Parameters Help	*angle*	// 회전하기 위한 각도
	x	// 회전하기 위한 중심축(X축)
	y	// 회전하기 위한 중심축(Y축)
	z	// 회전하기 위한 중심축(Z축)

glRotatef 함수를 사용한 객체의 회전(Rotation)

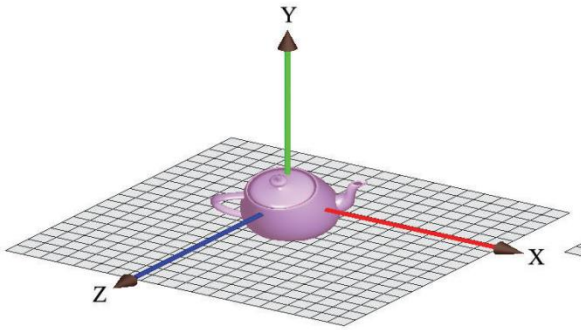
```
…
glRotatef(45.0, 0.0, 0.0, 1.0);
glutSolidTeapot(1.0);
…
```



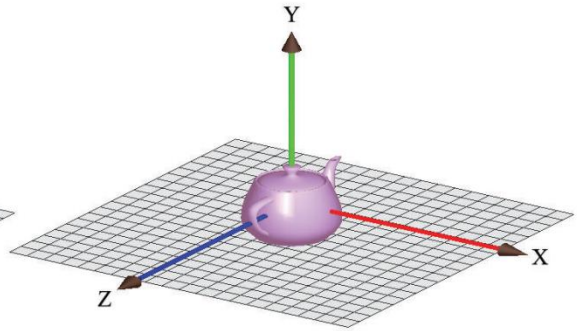
(A) glRotatef(0.0, 0.0, 0.0, 0.0);



(B) glRotatef(45.0, 0.0, 0.0, 1.0);



(C) glRotatef(45.0, 1.0, 1.0, 0.0);



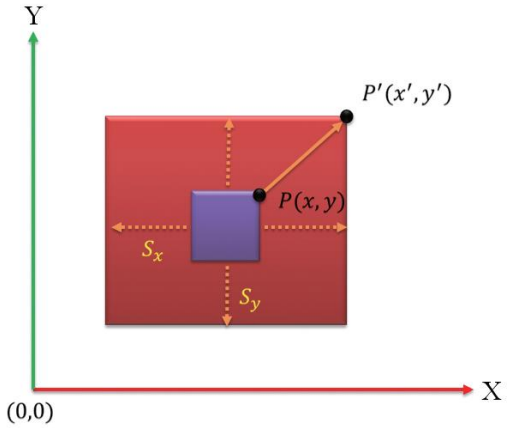
(D) glRotatef(90.0, 0.0, 1.0, 0.0);

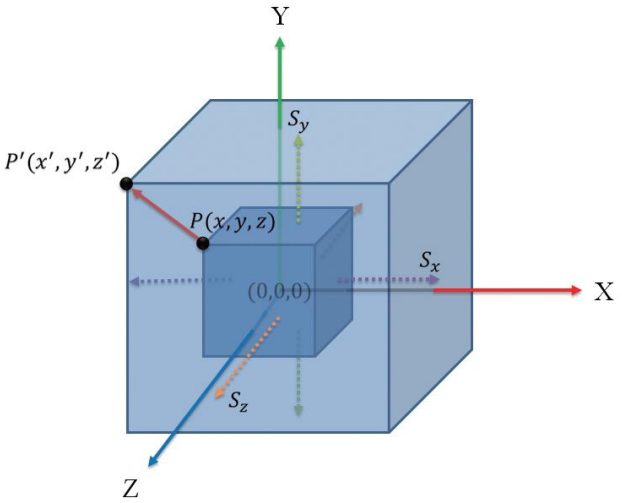
확대/축소(Scale)

- 2D 및 3D 공간에서의 크기 조절(Scale)

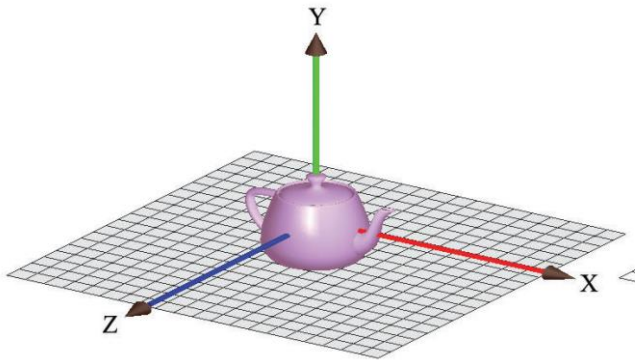
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \\ s_z \cdot z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cdot x \\ s_y \cdot y \\ s_z \cdot z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = S \cdot P$$

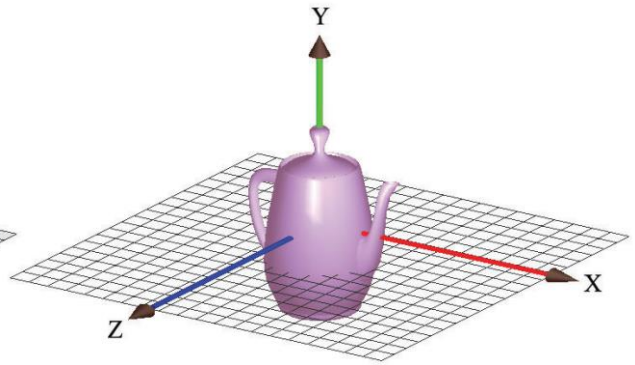




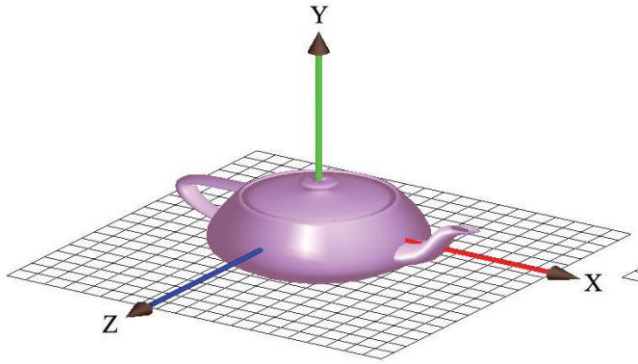
void glScalef (GLfloat *x*, GLfloat *y*, GLfloat *z*);	
Parameters Help	*x* // X축으로 크기 조절을 적용하기 위한 Scale 값 *y* // Y축으로 크기 조절을 적용하기 위한 Scale 값 *z* // Z축으로 크기 조절을 적용하기 위한 Scale 값



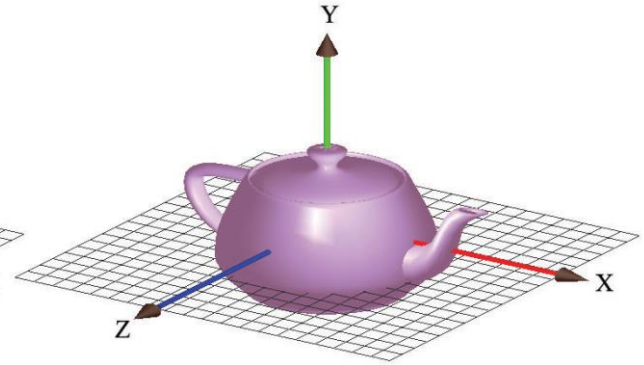
(A) glScalef(1.0, 1.0, 1.0);



(B) glScalef(1.0, 2.5, 1.0);

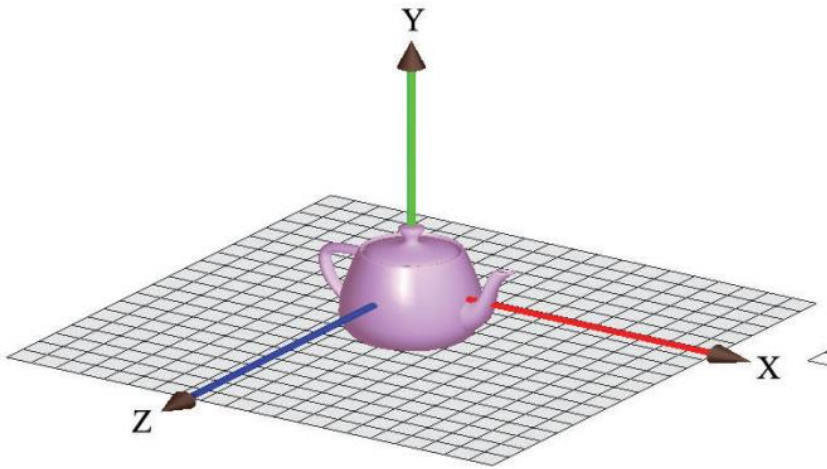


(C) glScalef(2.0, 1.0, 2.0);

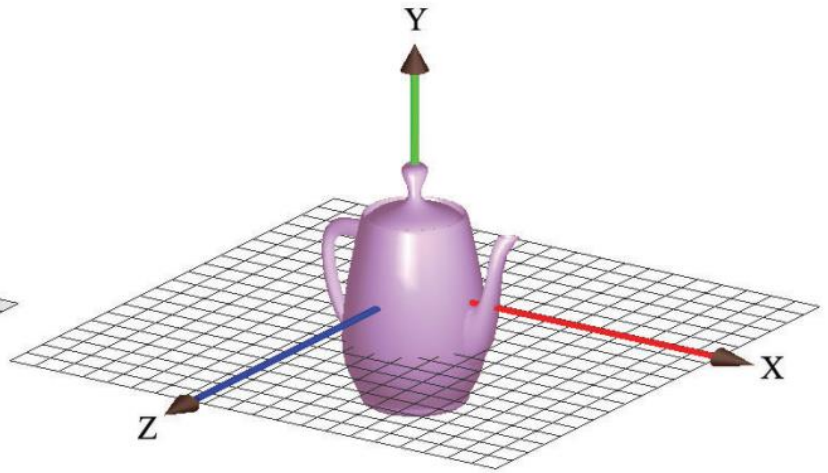


(D) glScalef(2.0, 2.0, 2.0);

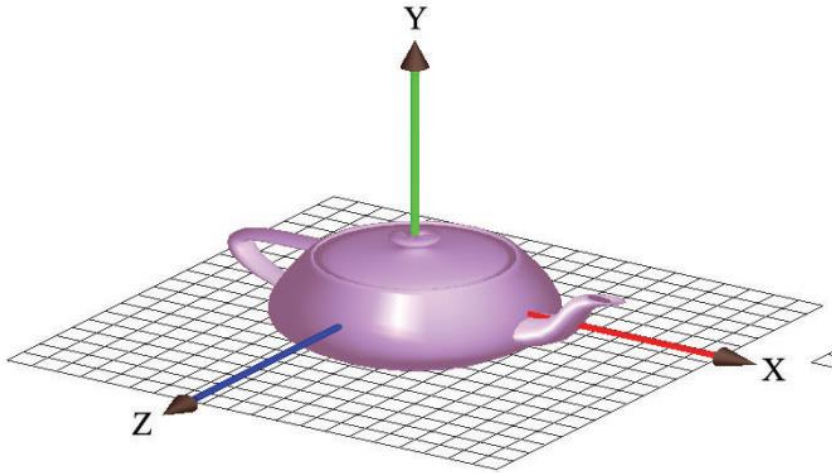
확대/축소(Scale)



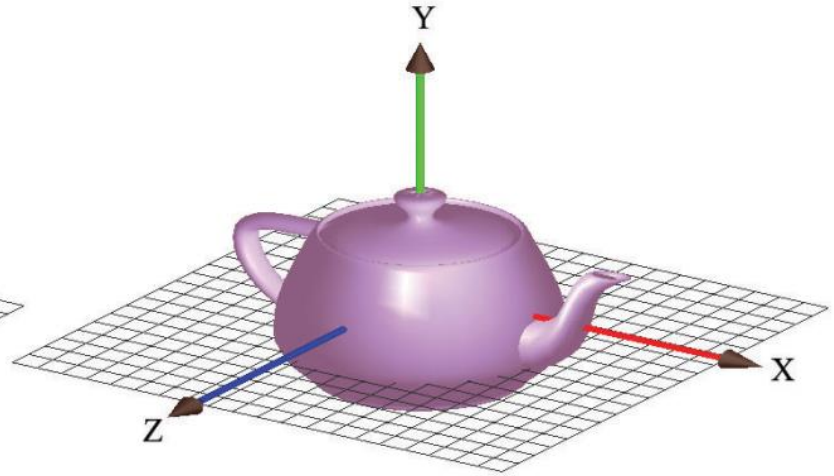
(A) glScalef(1.0, 1.0, 1.0);



(B) glScalef(1.0, 2.5, 1.0);



(C) glScalef(2.0, 1.0, 2.0);



(D) glScalef(2.0, 2.0, 2.0);

복합 변환 - Matrix의 연산 순서와 OpenGL 코드에서의 함수 호출 순서

Coordinate Transformation
[OpenGL Function]

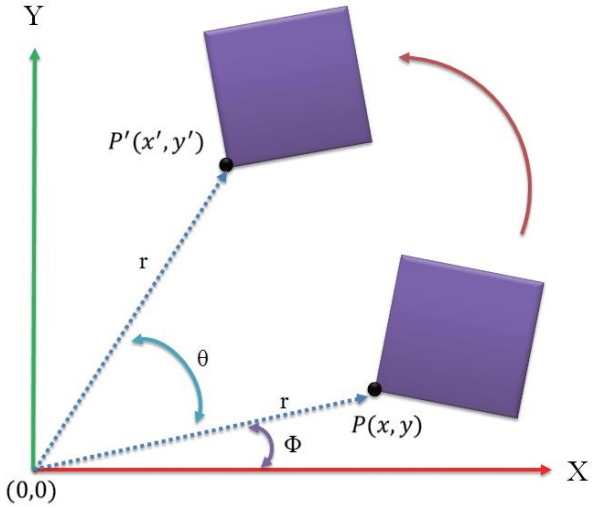
$$P' = I \bullet R \bullet T \bullet S \bullet P$$

Object Transformation

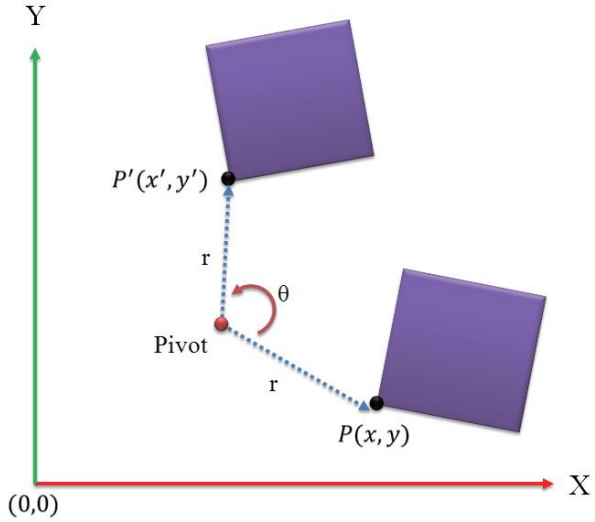
```
……
glLoadIdentity();
glRotate(135.0, 0.0, 1.0, 0.0);
glTranslatef(1.0, 0.0, 1.0);
glScale(1.0, 3.0, 1.0);
glVertex3f(1.0, 1.0, 1.0);
……
```

복합 변환 – 회전 변환의 기준점

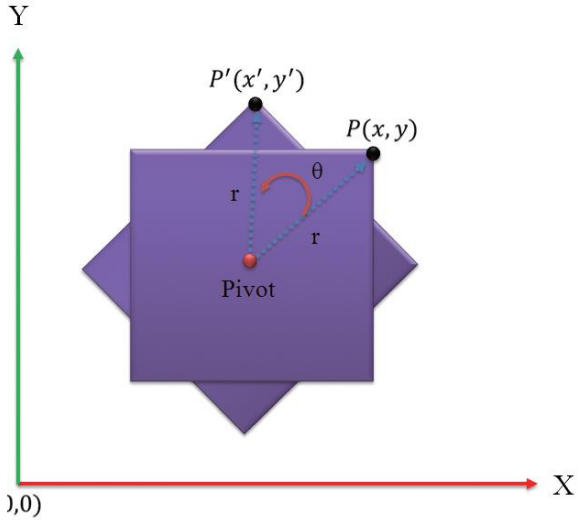
- 2D 공간에서 객체의 회전을 위한 기준점의 종류



(A) 기준점 - 원점

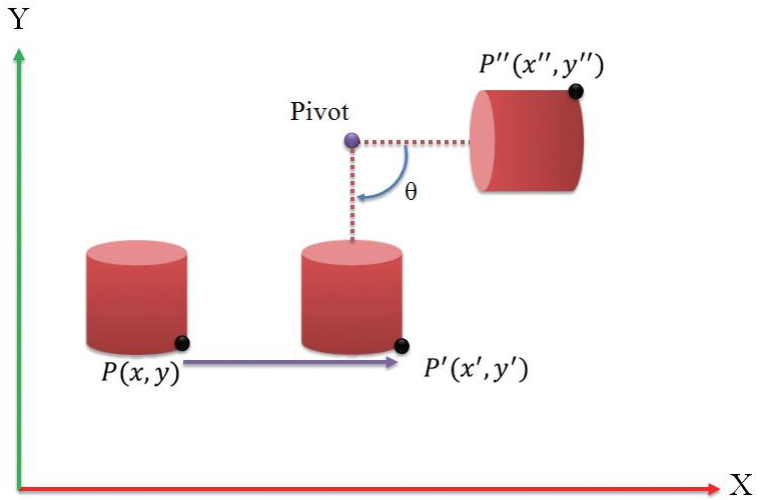


(B) 기준점 - 임의의 한 점(Point)

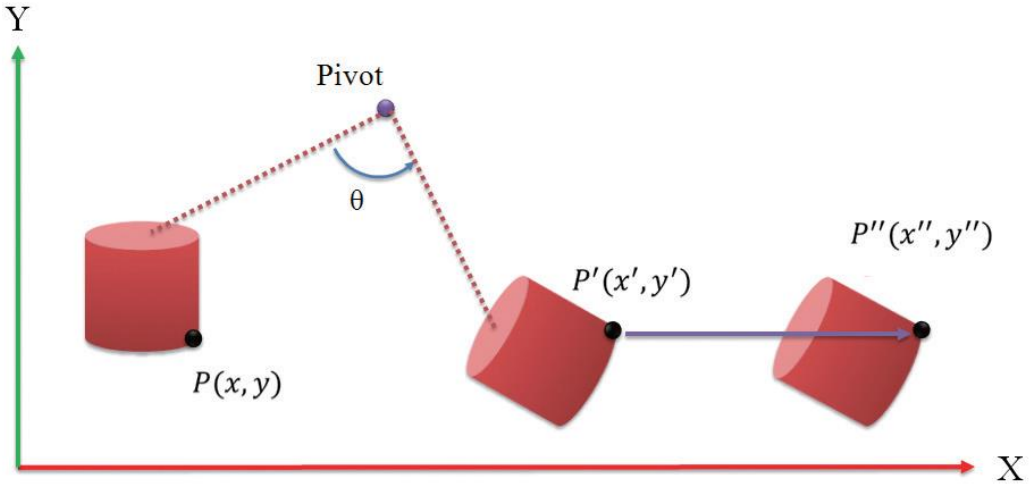


(C) 기준점 - 객체의 중심점

복합 변환 – 변환 순서에 따른 서로 다른 결과



(A) Translation → Rotation

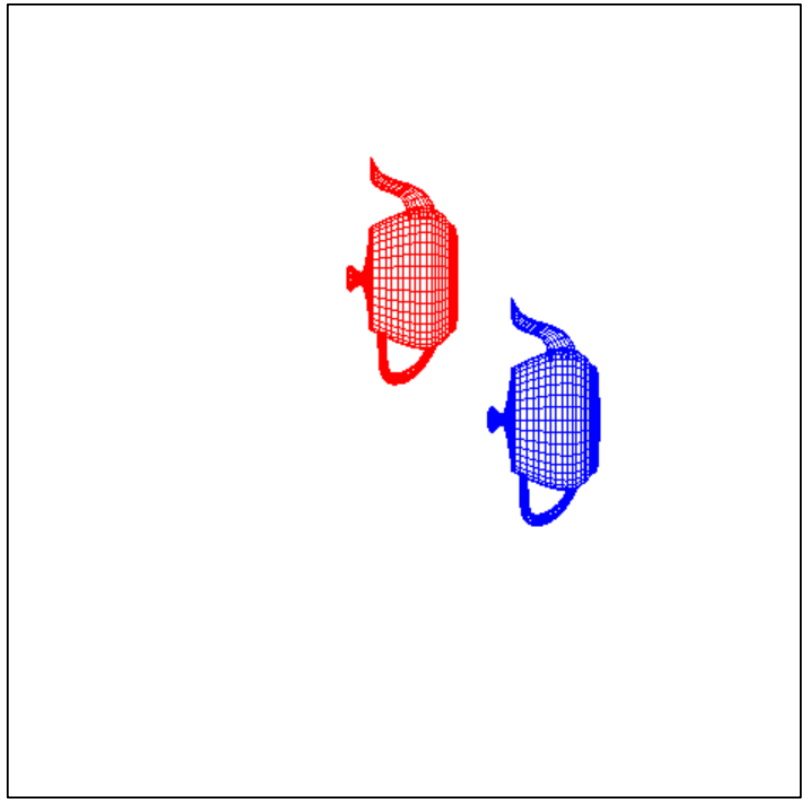


(B) Rotation → Translation

OpenGL에서의 변환 순서 비교

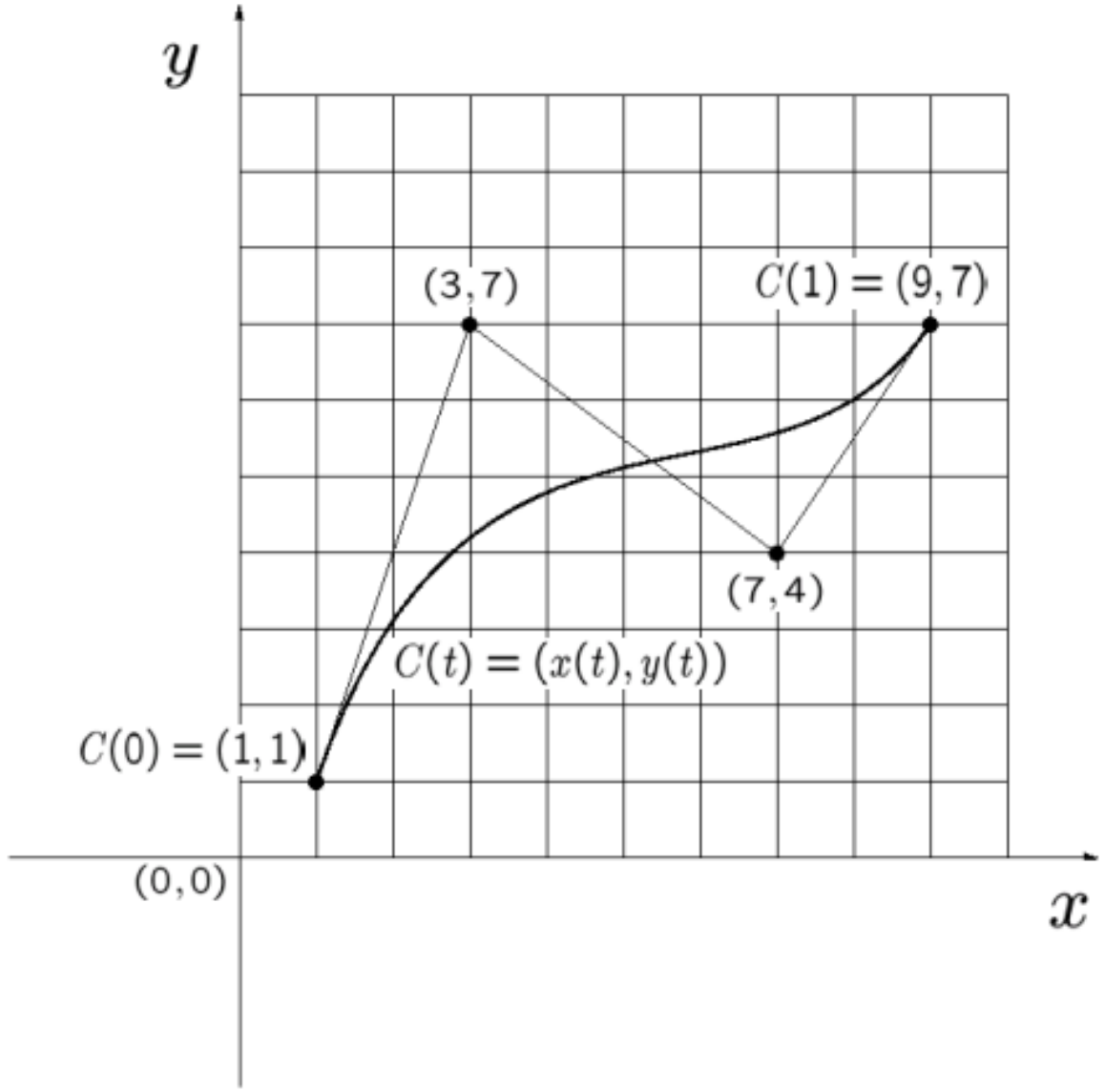
```
// ------------------------------------------------------------
    glMatrixMode(GL_MODELVIEW);
    // Rotate-Translate
    glLoadIdentity();
    glColor3f(1, 0, 0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    glTranslatef(1.0, 0.0, 0.0);
    displayGlutPrimitives();

    //Translate-Rotate
    glColor3f(0, 0, 1);
    glLoadIdentity();
    glTranslatef(1.0, 0.0, 0.0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    displayGlutPrimitives();
// ------------------------------------------------------------
```



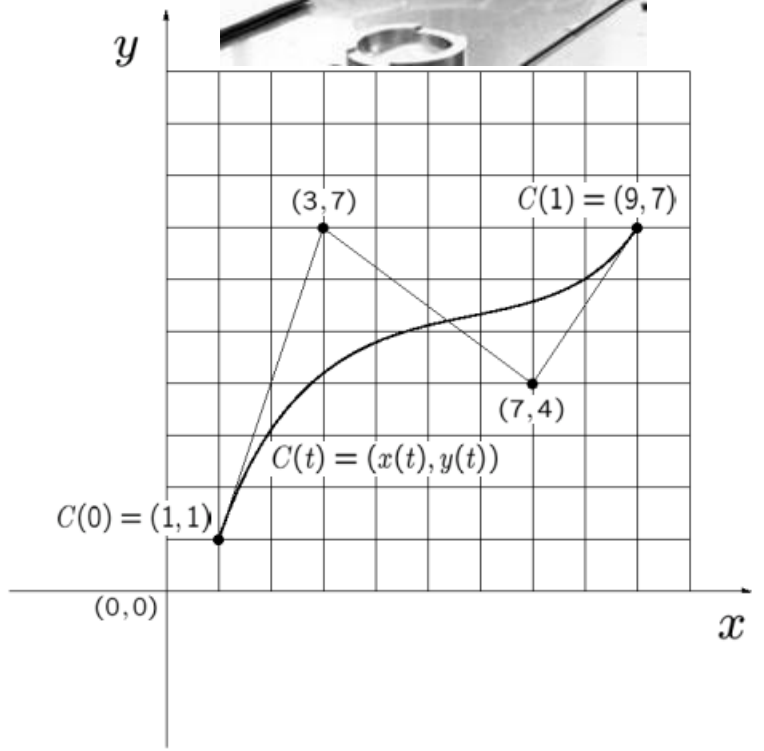
Curve

- 수식을 사용하여 표현
- 하나의 수식에 대해, 다양한 해상도로 표현 가능
- $C(t)$: 매개변수 곡선, 새로운 매개변수 t 사용



베지어(Bezier) 곡선

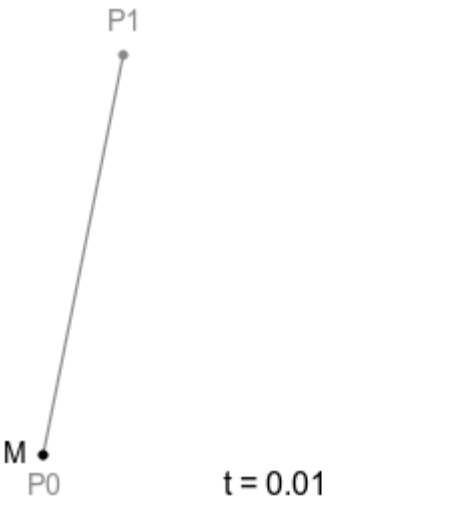
- **가장 널리 사용되는 벡터 그래픽 곡선**
  - 프랑스의 수학자 Paul de Casteljau가 처음 곡선 형태로 사용
  - 르노의 기술자 피에르 베지어가 널리 사용
- **$n(n \geq 2)$개의 조절점(Control Point)로 정의되는 매개변수 곡선**
- **n개의 점으로 $(n-1)$차 베지어 곡선 생성 가능**

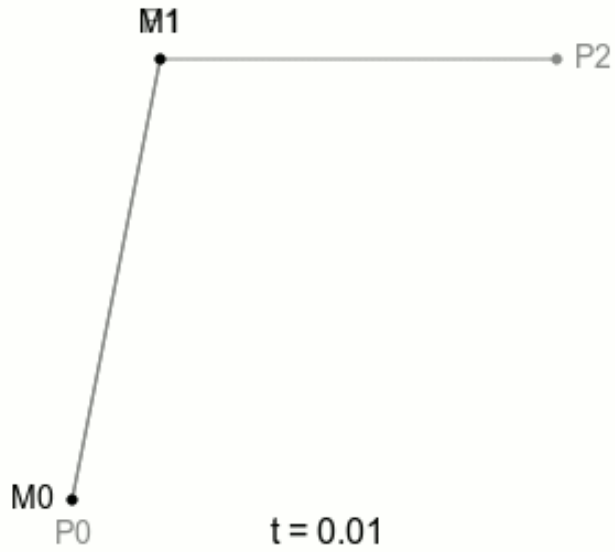


베지어 곡선

- **1차 베지어 곡선**
  - 2개의 Control Points(P_n)로 생성
  - $C(t) = (1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1$
- **2차 베지어 곡선**
  - 3개의 Control Points(P_n)로 생성
  - $C(t) = (1-t)[(1-t)\boldsymbol{P}_0 + t\boldsymbol{P}_1] + t[(1-t)\boldsymbol{P}_1 + t\boldsymbol{P}_2]$

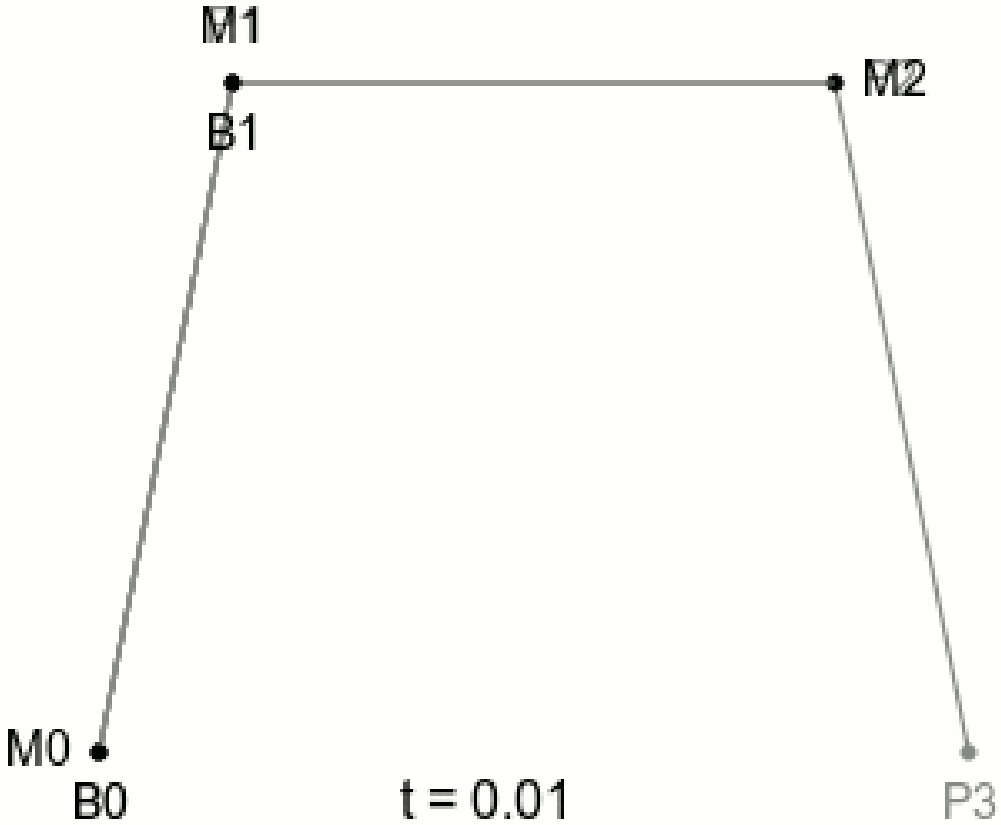
    $= (1-t)^2\boldsymbol{P}_0 + 2(1-t)t\boldsymbol{P}_1 + t^2\boldsymbol{P}_2$
  - Quadratic Bezier Curve
  - TrueType Font 등 사용





3차 베지어 곡선

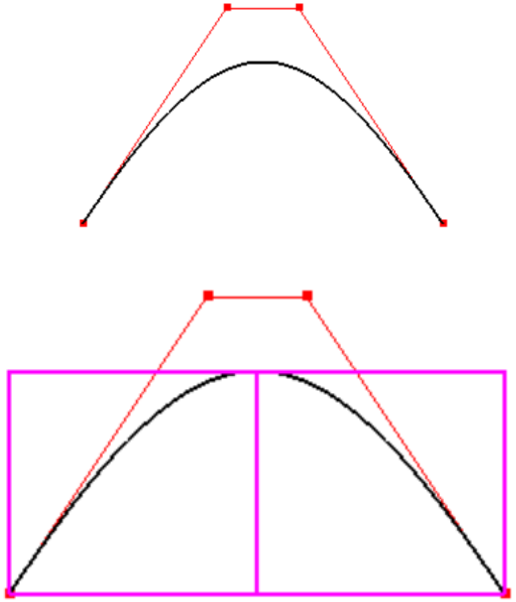
- $C(t) = (1-t)^3\boldsymbol{P}_0 + 3(1-t)^2 t\boldsymbol{P}_1 + 3(1-t)t^2\boldsymbol{P}_2 + t^3\boldsymbol{P}_3$
- Cubic Bezier curve
- PostScript, Metafont, GIMP 등 사용

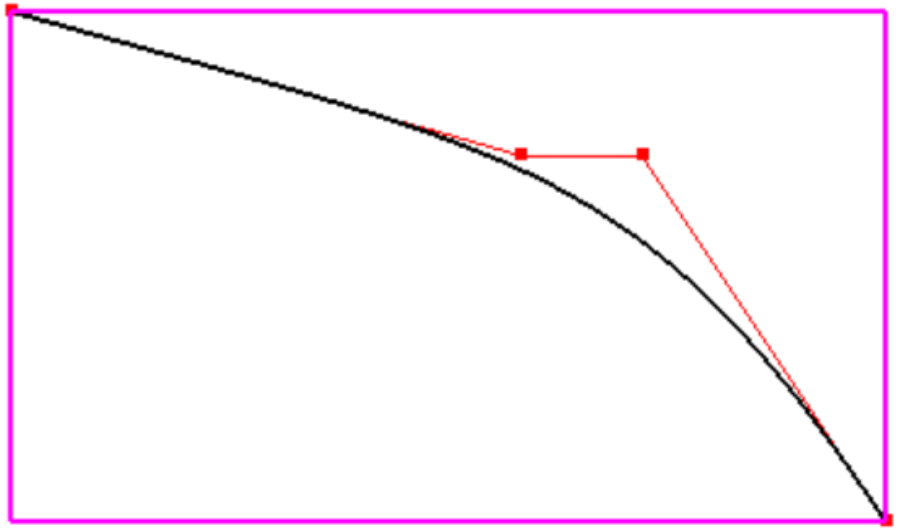


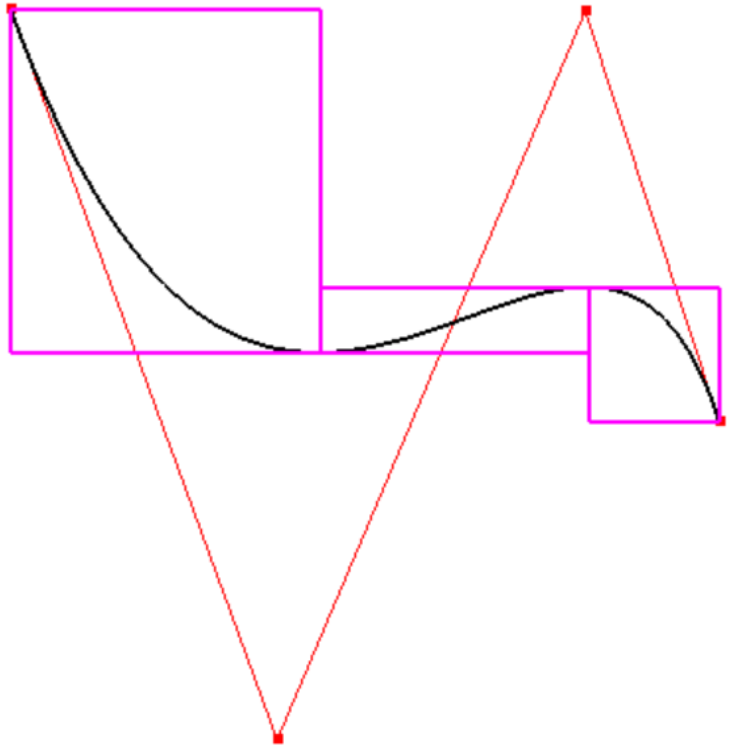
베지어 곡선과 친해지기

[기본과제] 4점

- Control Point를 하나 더 임의의 지점에 추가하여 3차 베지어 커브로 생성하기 (1점)
- 베지어 곡선를 구성하는 선분들의 기울기(= y변화량 / x변화량)와 std::vector를 활용하여 단조증가, 단조감소 영역으로 구분하기 (1점)
- 구분된 곡선들을 Axis-Aligned Bounding Box(AABB)로 감싸기 (2점)
- 마감기한 : 3/29 (금) 23:59







Any Questions?