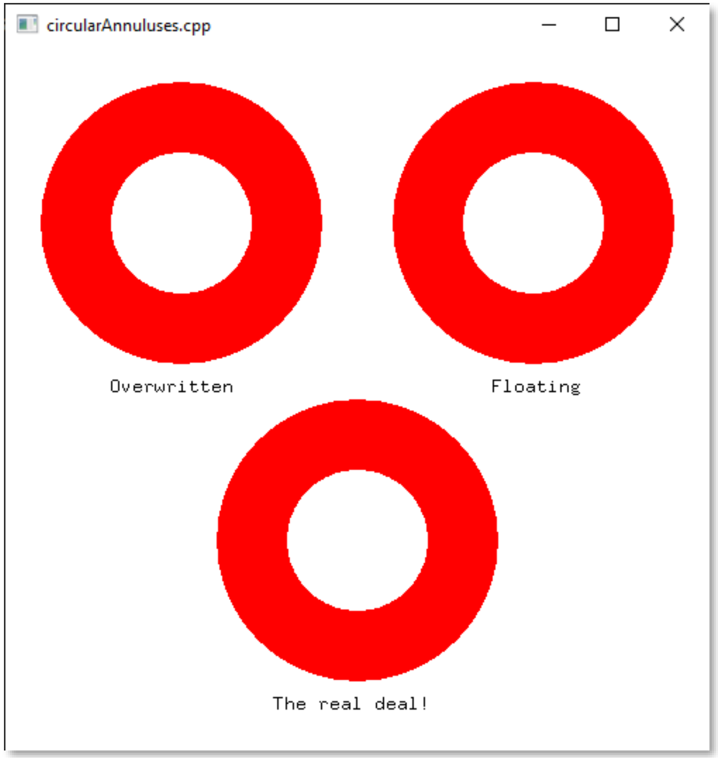
COMPUTER GRAPHICS

3D Graphics with OpenGL

Dr. Zhonggui Chen
School of Informatics, Xiamen University
http://graphics.xmu.edu.cn

Example: circularAnnuluses.cpp

- \ExperimenterSource\Chapter2\CircularAnnuluses\circularAnnuluses.cpp



Example: circularAnnuluses.cpp

- Upper left circular annulus

```
// the white disc overwrites the red disc.
glColor3f(1.0, 0.0, 0.0); // red
drawDisc(20.0, 25.0, 75.0, 0.0);
glColor3f(1.0, 1.0, 1.0); // white
drawDisc(10.0, 25.0, 75.0, 0.0);
```

Example: circularAnnuluses.cpp

- Upper right circular annulus

```
// the white disc is in front of the red disc blocking it.
glEnable(GL_DEPTH_TEST); // Enable depth testing.
glColor3f(1.0, 0.0, 0.0);
drawDisc(20.0, 75.0, 75.0, 0.0);
glColor3f(1.0, 1.0, 1.0);
drawDisc(10.0, 75.0, 75.0, 0.5); // Compare this z-value
with that of the red disc.
glDisable(GL_DEPTH_TEST); // Disable depth testing.
```

Example: circularAnnuluses.cpp

- Lower circular annulus

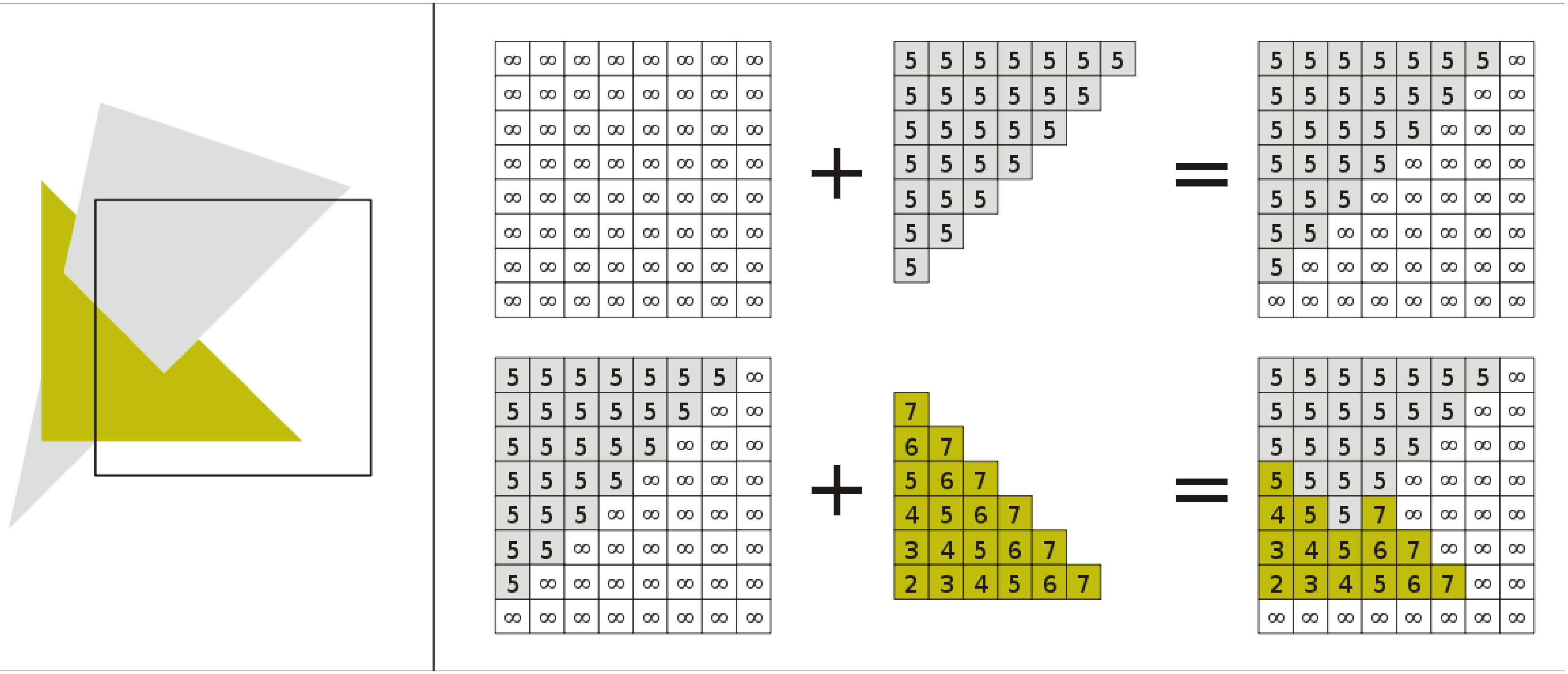
```
// with a true hole.
if (isWire) glPolygonMode(GL_FRONT, GL_LINE);
else glPolygonMode(GL_FRONT, GL_FILL);
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_TRIANGLE_STRIP);
for (i = 0; i <= N; ++i)
{
    angle = 2 * PI * i / N;
    glVertex3f(50 + cos(angle) * 10.0,
       30 + sin(angle) * 10.0, 0.0);
    glVertex3f(50 + cos(angle) * 20.0,
       30 + sin(angle) * 20.0, 0.0);
}
glEnd();
```

Example: circularAnnuluses.cpp

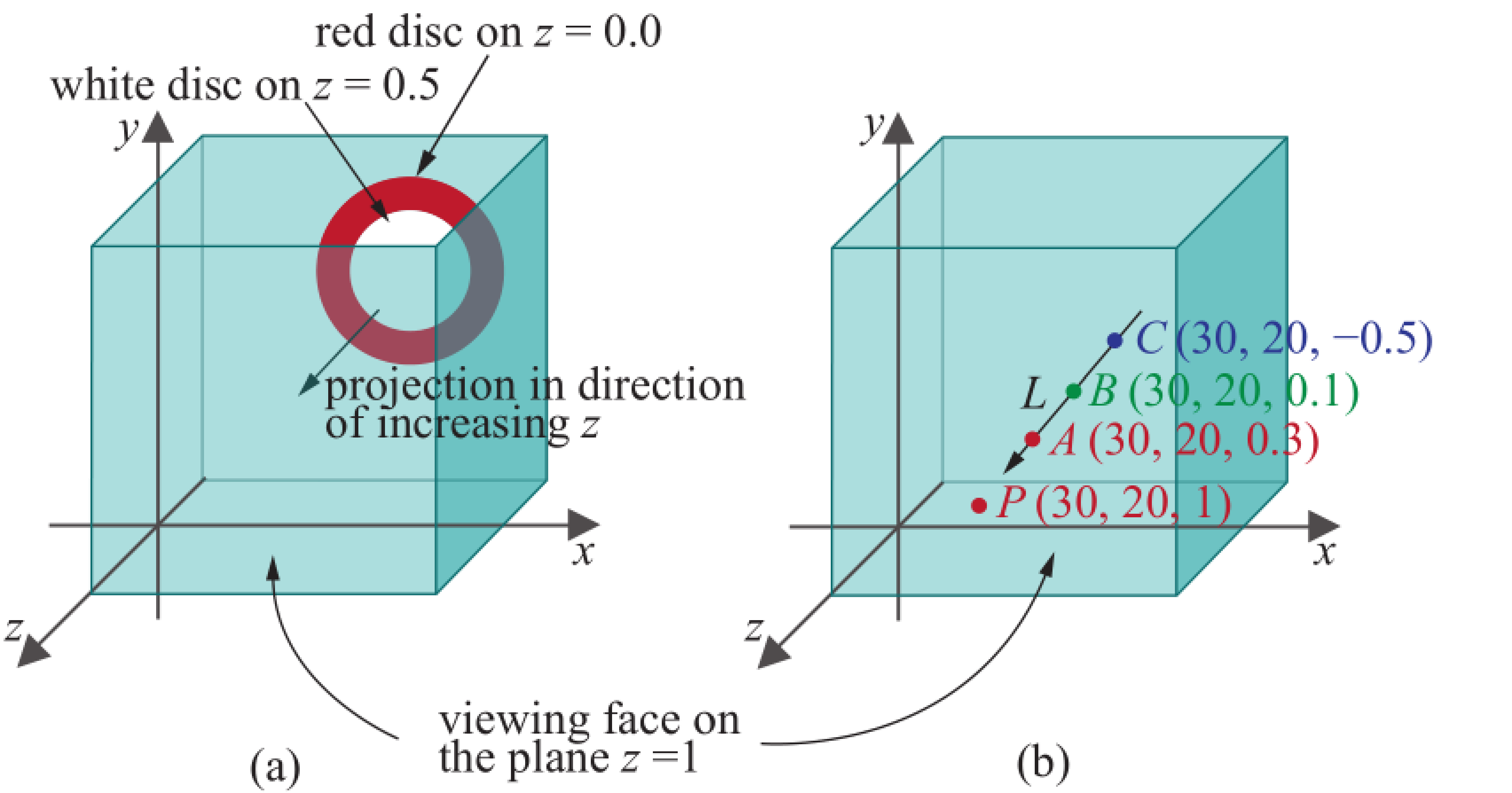
- Interchange in circularAnnuluses.cpp the drawing orders of the red and white discs – i.e., the order in which they appear in the code – in either of the top two annuluses.
- Which one is affected?
- Why?

Z-Buffer

- Add extra depth channel to image
- Write Z values when writing pixels
- Test Z values before writing



Z-Buffer



red disc on z = 0.0
white disc on z = 0.5
y
x
z
projection in direction
of increasing z
viewing face on
the plane z =1
(a)
y
x
z
C (30, 20, −0.5)
L
B (30, 20, 0.1)
A (30, 20, 0.3)
P (30, 20, 1)
(b)

Z-Buffer

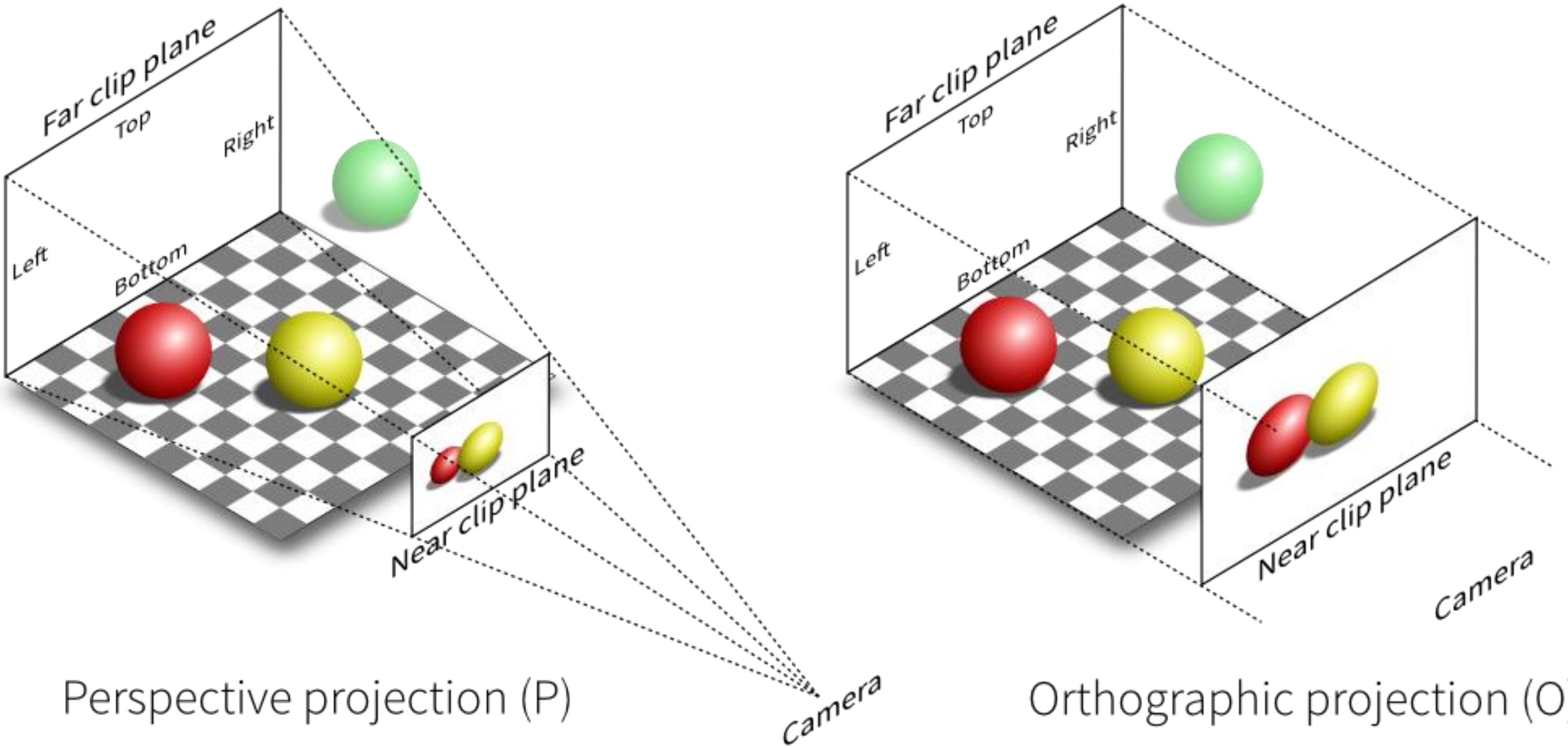
A 3D scene

Z-buffer representation

Enabling Depth Testting in OpenGL

- glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB| GLUT_DEPTH) in main() causes the depth buffer to be initialized.
- glClear(GL_ COLOR_BUFFER BIT | GL_DEPTH_BUFFER_BIT) in the drawScene() routine causes the depth buffer to be cleared.
- glEnable(GL_DEPTH_TEST) in the drawScene() routine turns hidden surface removal on.
  - The complementary command is glDisable(GL_DEPTH_TEST)

Projection transformation



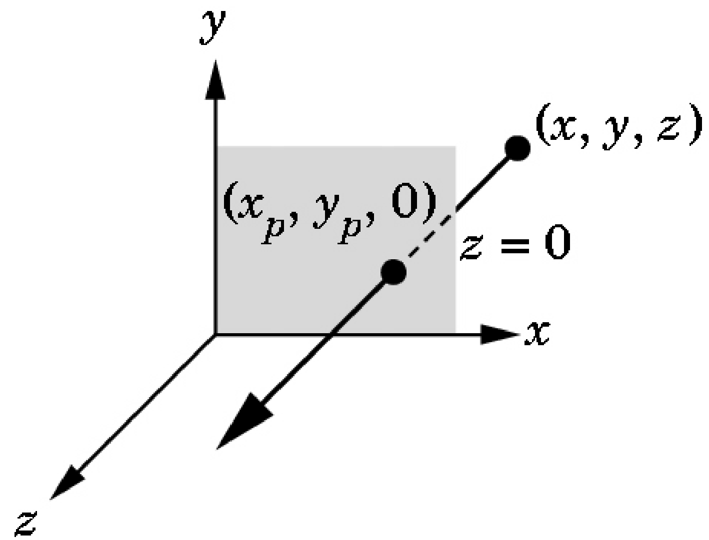
Perspective projection (P)

Orthographic projection (O)

Orthographic Projection Matrix

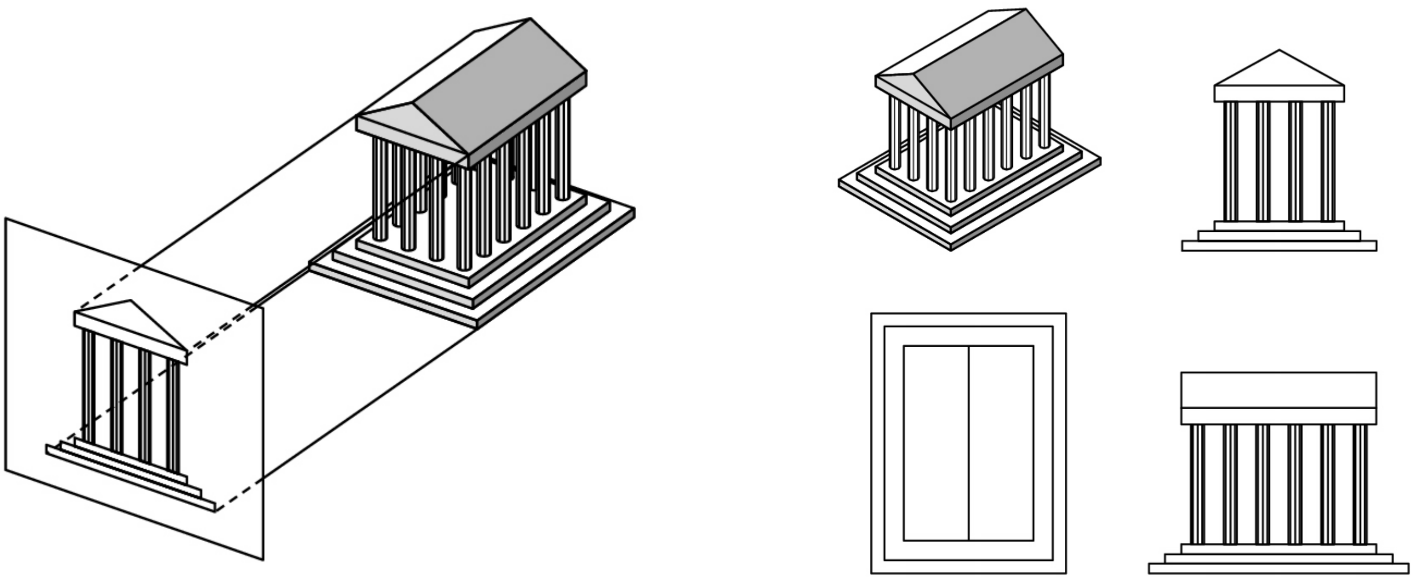
- Original point: (x, y, z, 1)
- Projection onto z = 0 plane
- Projected point: $x_p = x$, $y_p = y$, $z_p = 0$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

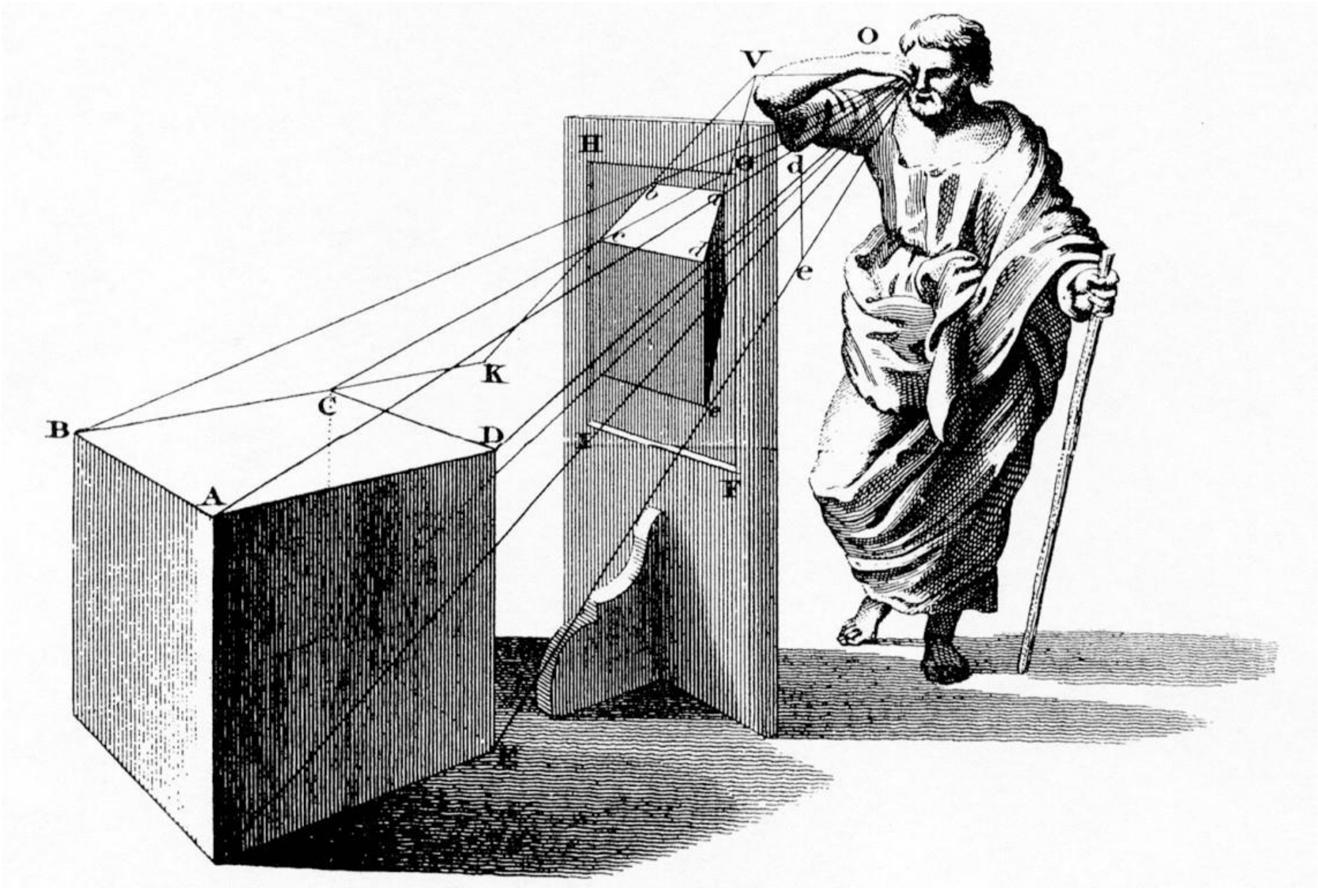


Orthographic Projection

- Simple, but not realistic
- Used in architectural design

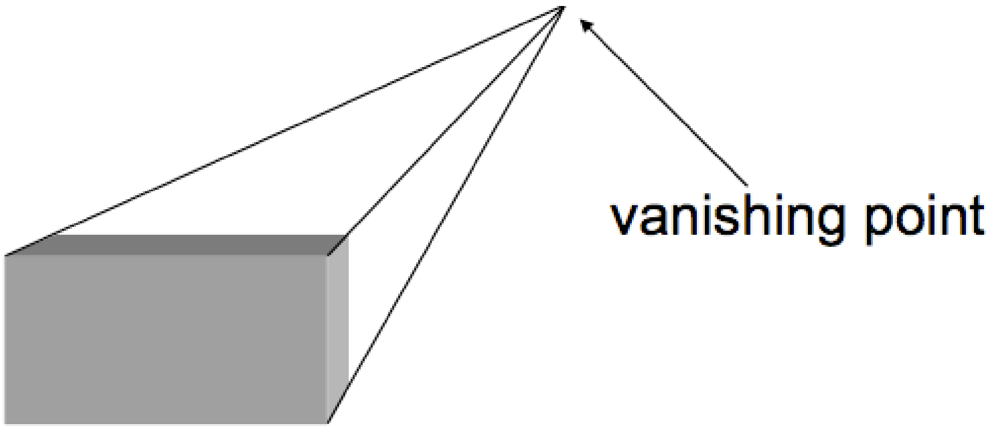


Perspective Projection

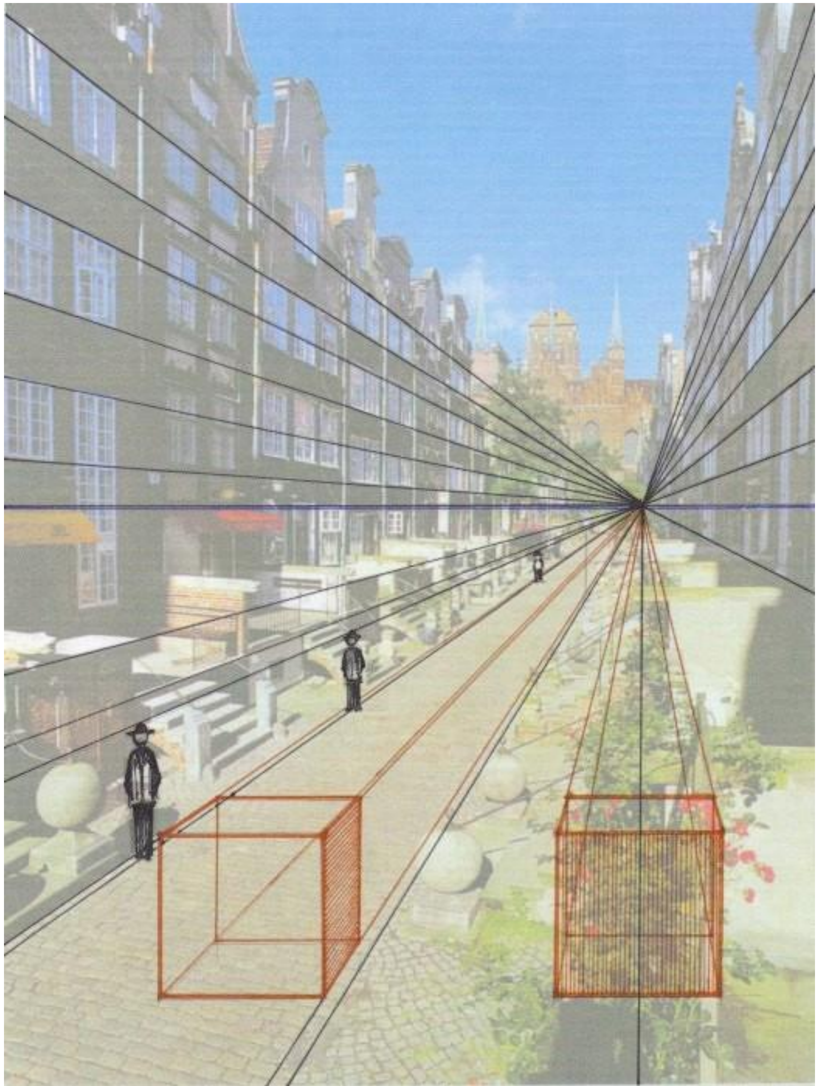


Vanishing Points

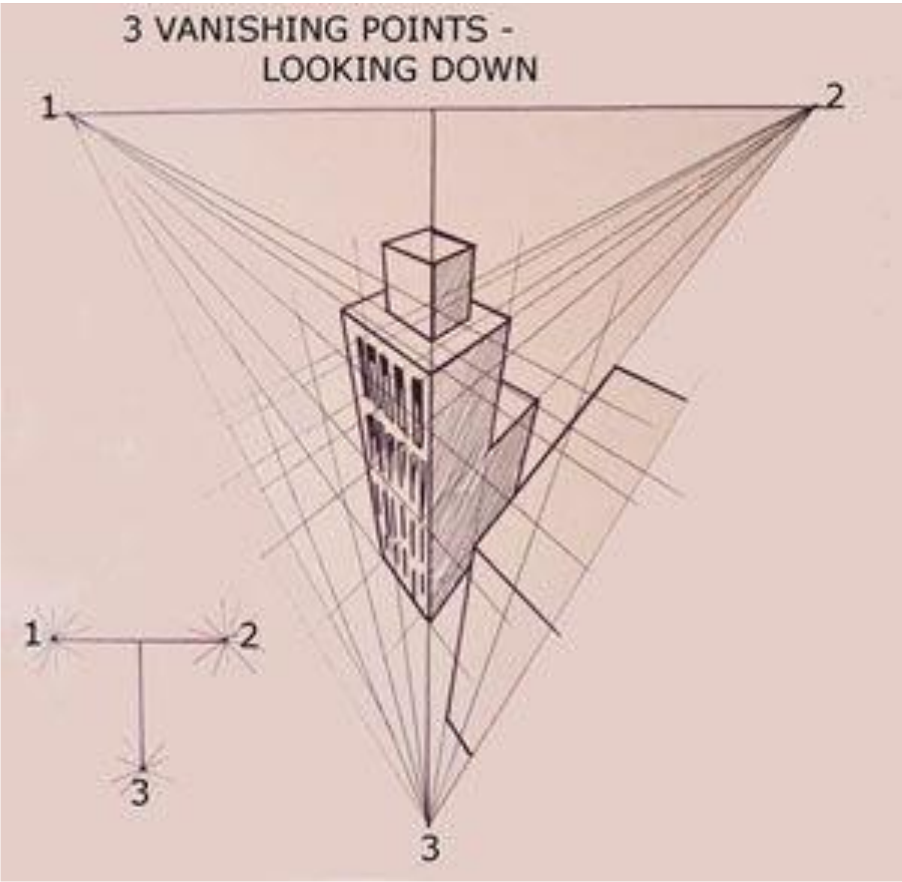
- In perspective projection, parallel lines (parallel in the scene) appear to converge to a single point
  - This is called the vanishing point



Vanishing Points

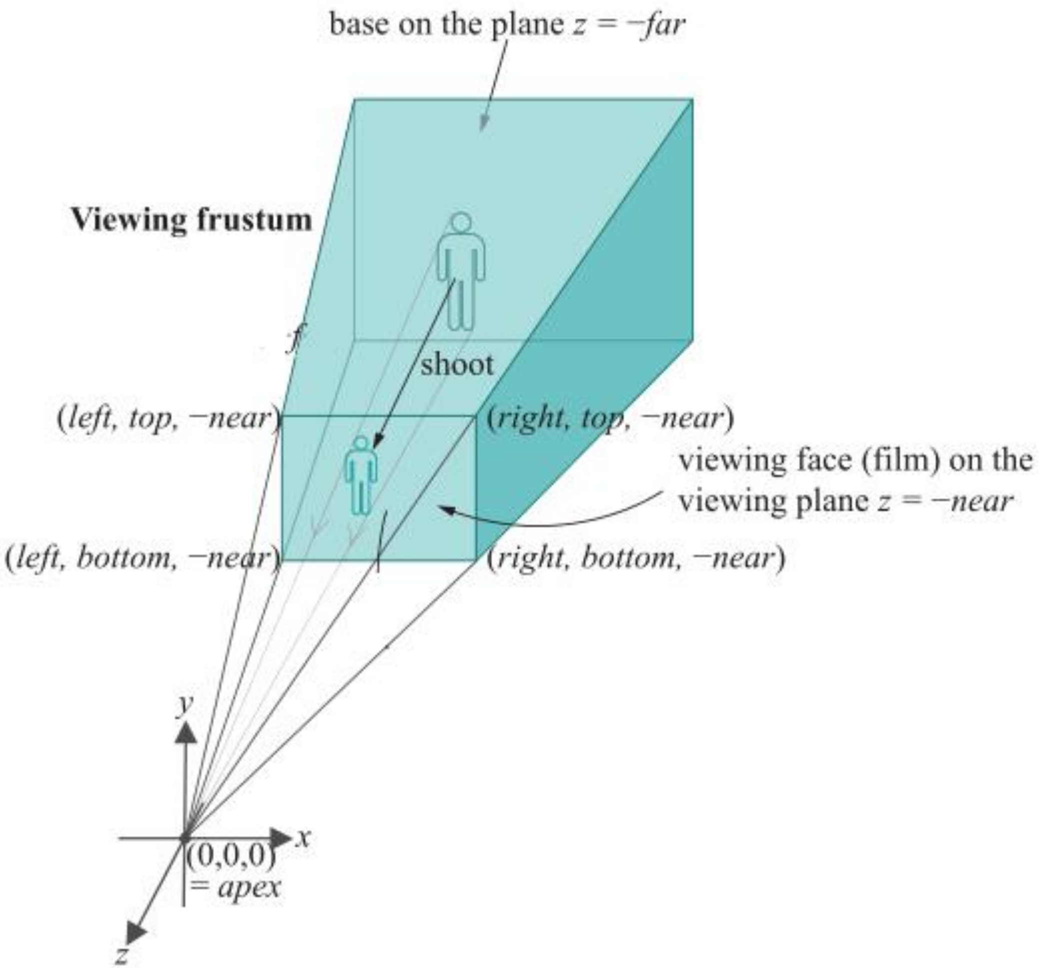


Vanishing Points



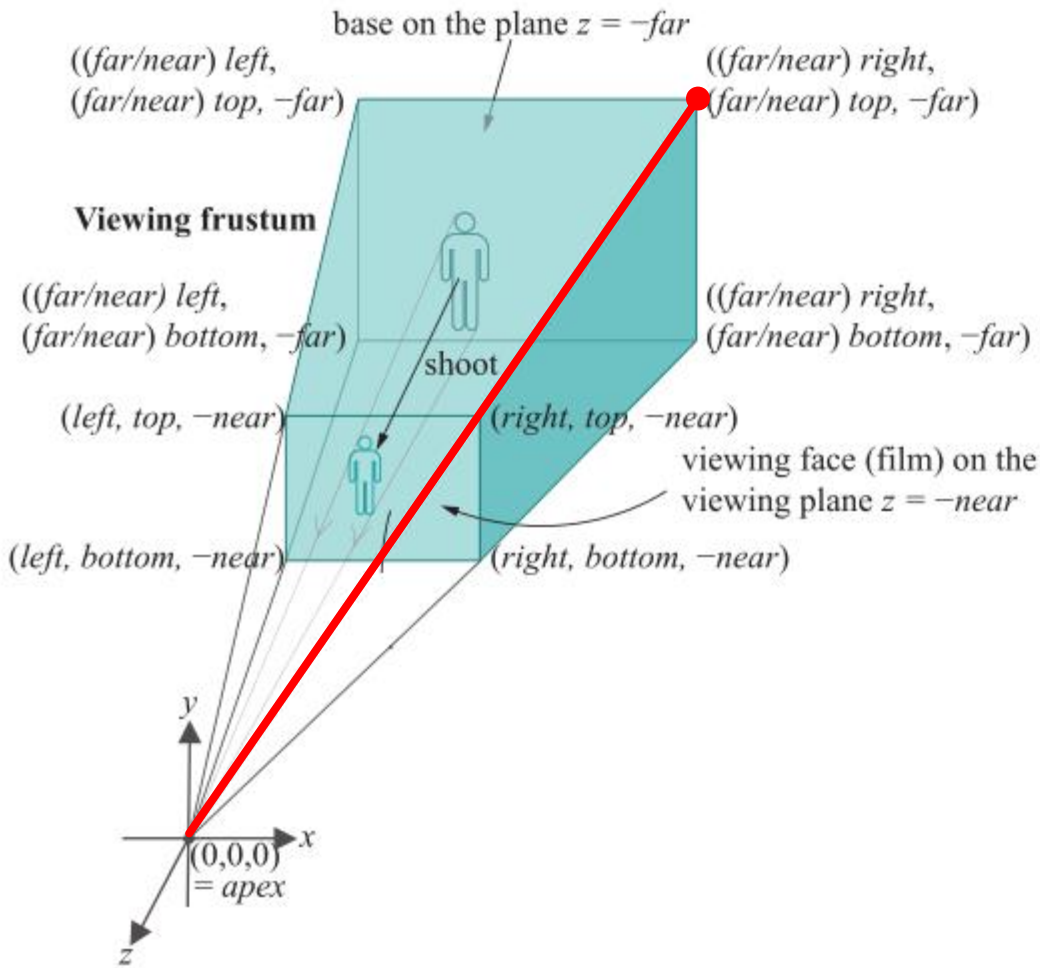
Perspective Projection in OpenGL

- glFrustum(left , right , bottom , top, near , far)



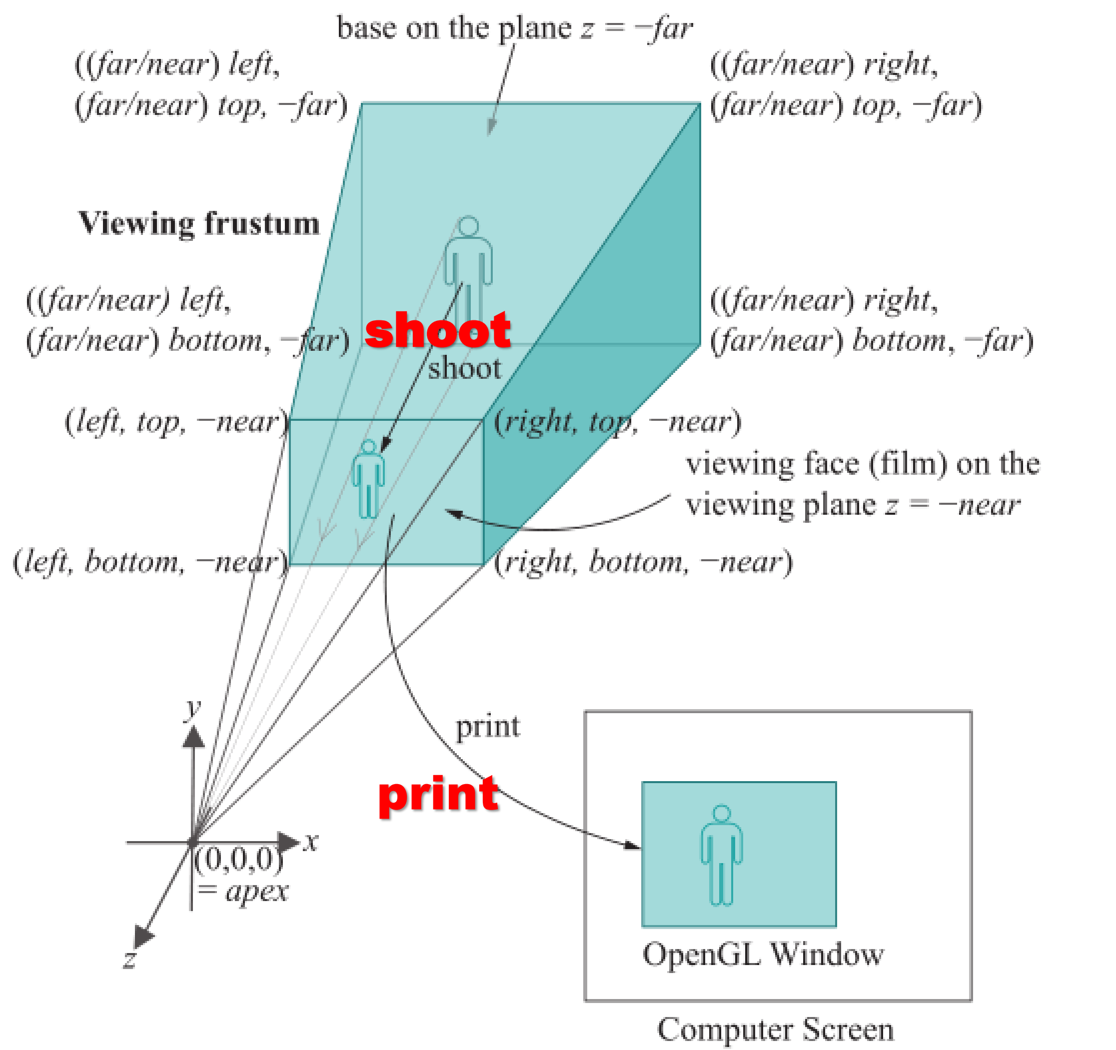
Perspective Projection in OpenGL

- glFrustum(left , right , bottom , top , near , far)



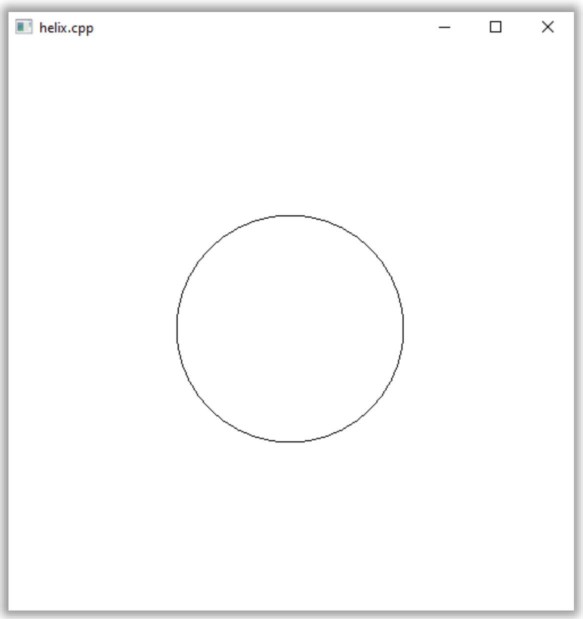
Perspective Projection in OpenGL

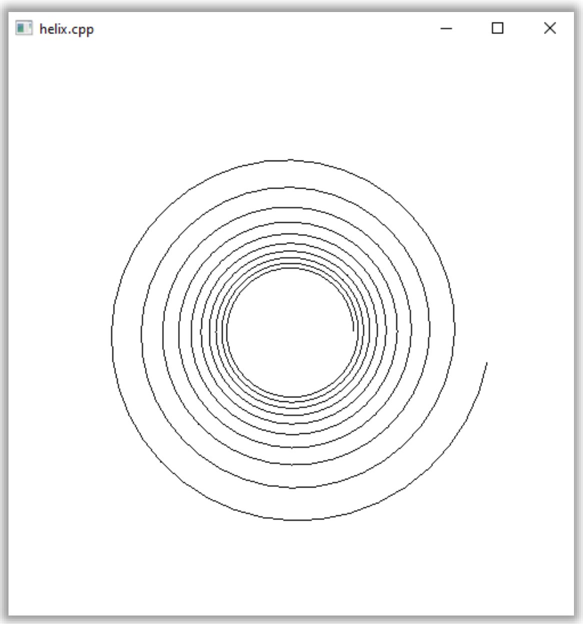
- Rendering sequence
  - Shoot and print



Example： helix.cpp

- A helix coiling up the z-axis: glVertex3f(R * cos(t), R * sin(t), t - 60.0);
- Replace glOrtho(-50.0, 50.0, -50.0, 50.0, 0.0, 100.0)
  with glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0)





Example： helix.cpp

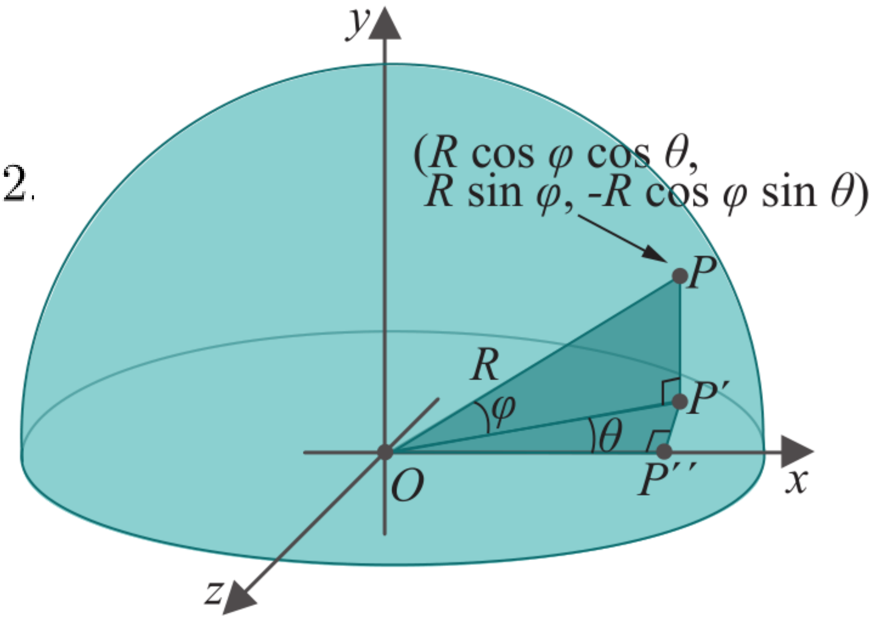
- **Predict the change in the display caused by the change in the frustum：**
- Move the back face back:
  - glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 120.0)
- Move the front face back:
  - glFrustum(-5.0, 5.0, -5.0, 5.0, 10.0, 100.0)
- Move the front face forward:
  - glFrustum(-5.0, 5.0, -5.0, 5.0, 2.5, 100.0)
- Make the front face bigger:
  - glFrustum(-10.0, 10.0, -10.0, 10.0, 5.0,100.0)

Approximating Curved Objects

- Hemisphere of radius R , centered at the origin O, with its circular base lying on the xz -plane

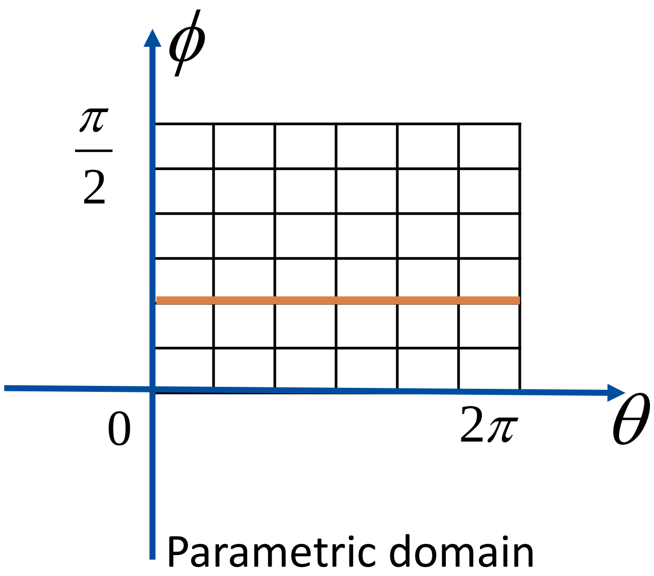
$$(R\cos\phi\cos\theta,\ R\sin\phi,\ -R\cos\phi\sin\theta)$$

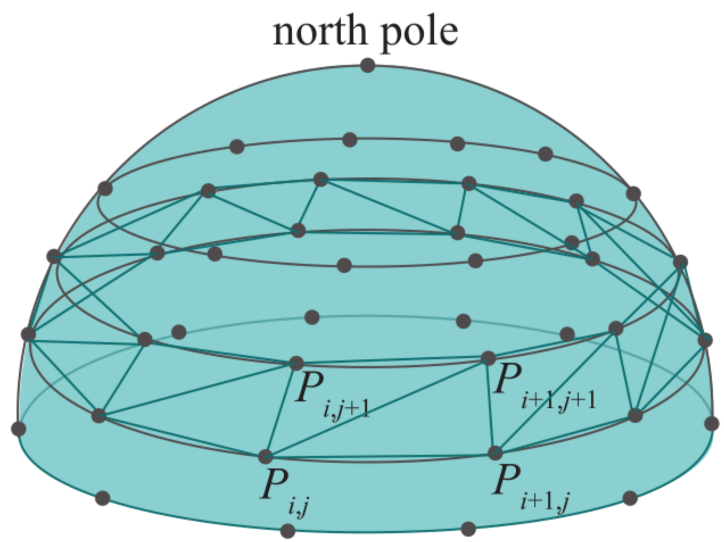
θ is $0 \le \theta \le 2\pi$ and of ϕ is $0 \le \phi \le \pi/2$.



Approximating Curved Objects

- Sample the hemisphere and approximate the circular band between each pair of adjacent latitudes with a triangle strip

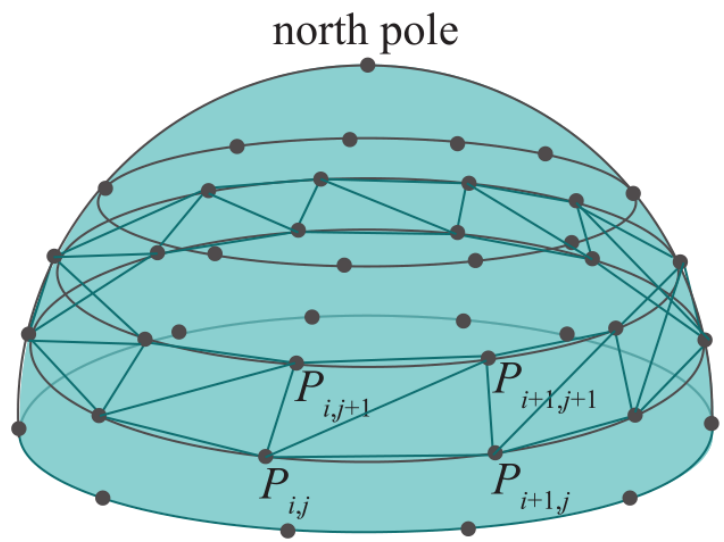




Approximating Curved Objects

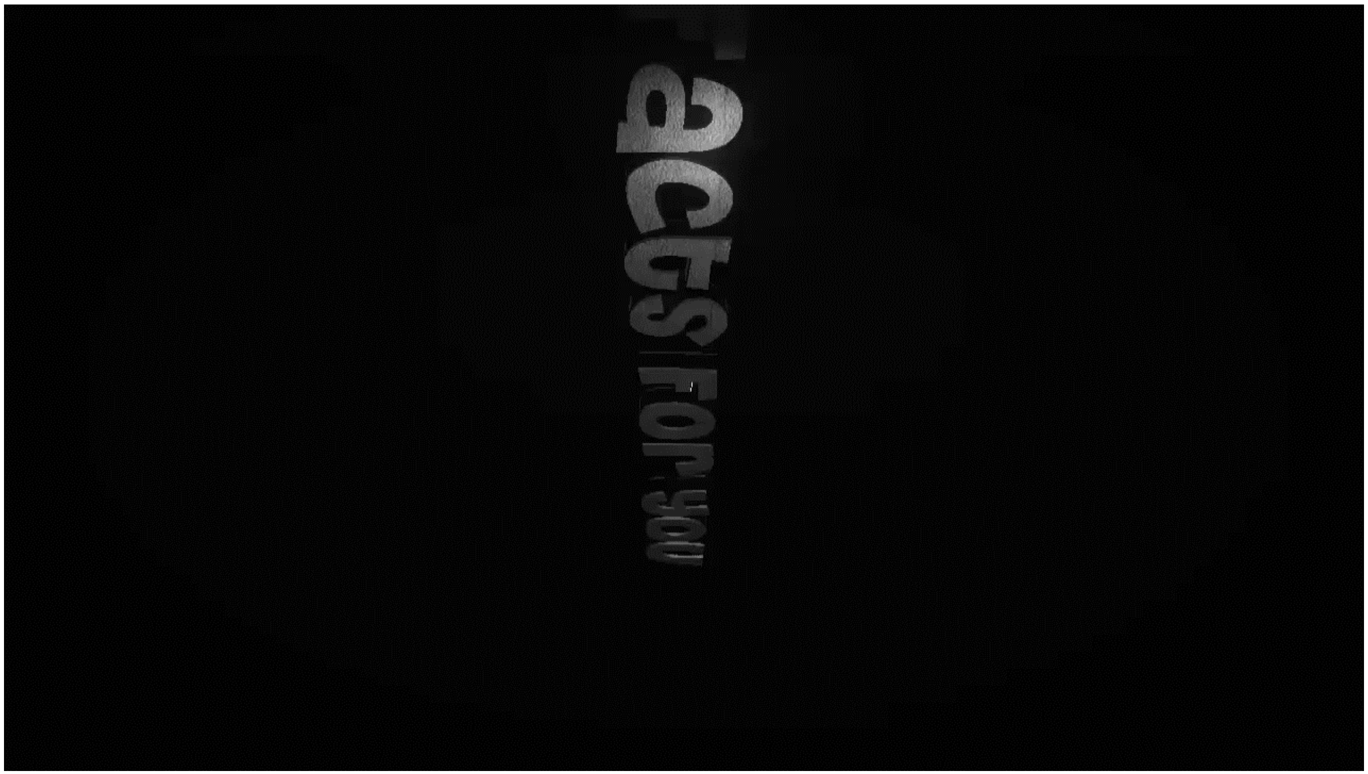
- Sample the hemisphere and approximate the circular band between each pair of adjacent latitudes with a triangle strip

```
for(j = 0; j < q; j++)
{
   // One latitudinal triangle strip.
   glBegin(GL_TRIANGLE_STRIP);
      for(i = 0; i <= p; i++)
      {
         glVertex3f(R * cos((float)(j+1)/q * PI/2.0) *
                         cos(2.0 * (float)i/p * PI),
                    R * sin((float)(j+1)/q * PI/2.0),
                    -R * cos((float)(j+1)/q * PI/2.0) *
                         sin(2.0 * (float)i/p * PI));
         glVertex3f(R * cos((float)j/q * PI/2.0) *
                         cos(2.0 * (float)i/p * PI),
                    R * sin((float)j/q * PI/2.0),
                    -R * cos((float)j/q * PI/2.0) *
                         sin(2.0 *(float)i/p * PI));
      }
   glEnd();
}
```



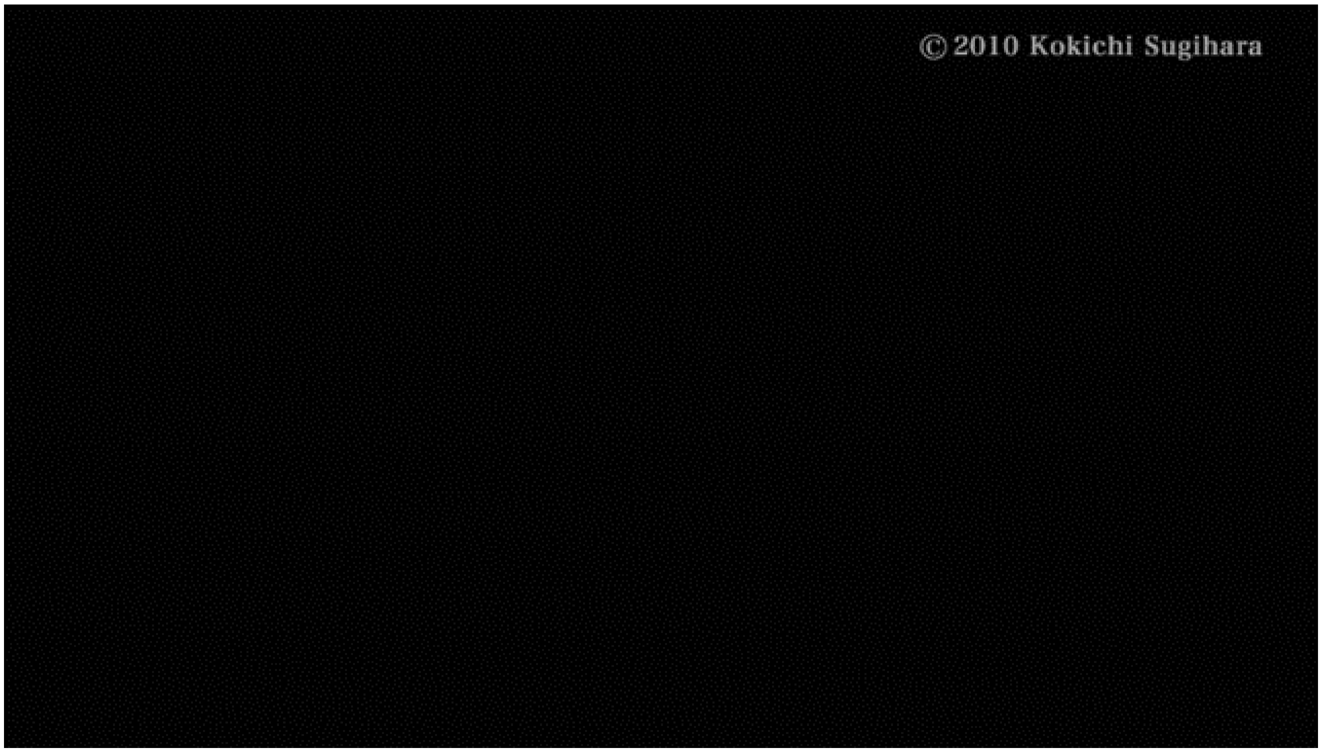
Interesting applications

- 3D Street Art



Interesting applications

- Impossible motions

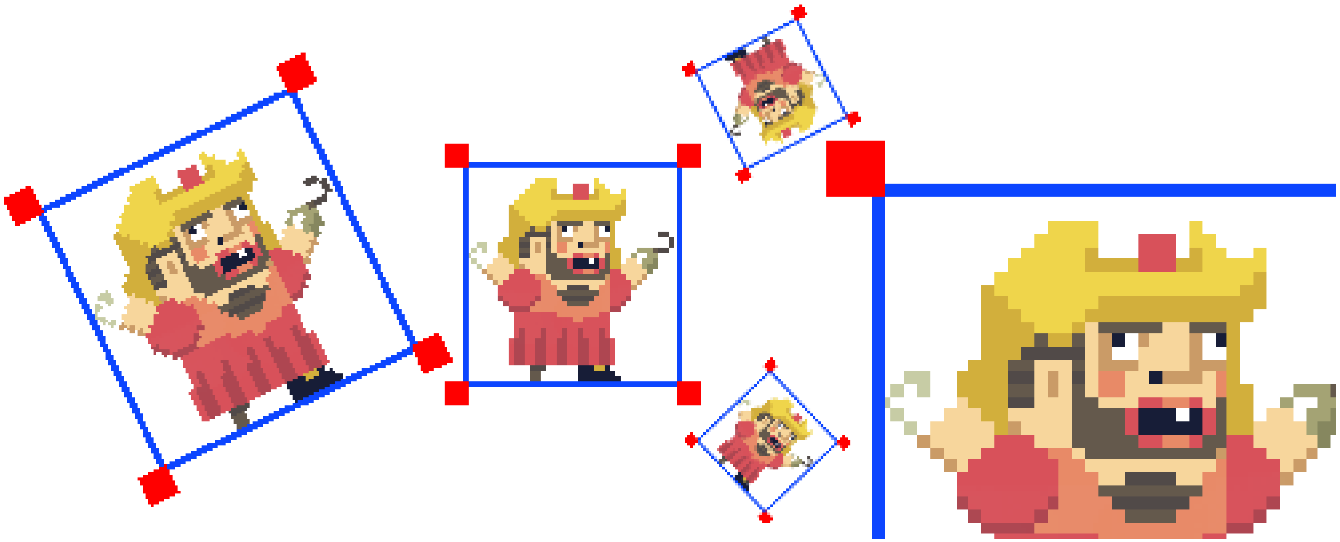


3D Graphics with OpenGL

-- Transformations

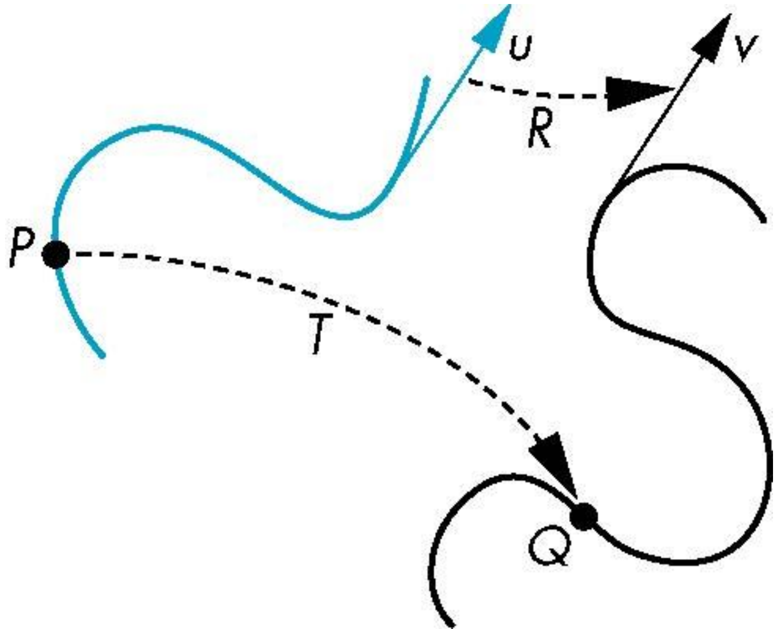
Transformations

Basics

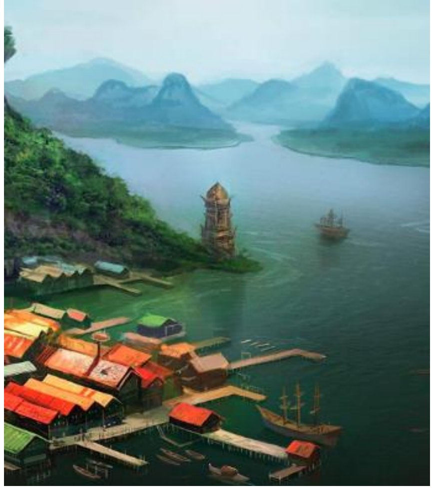


Introduction

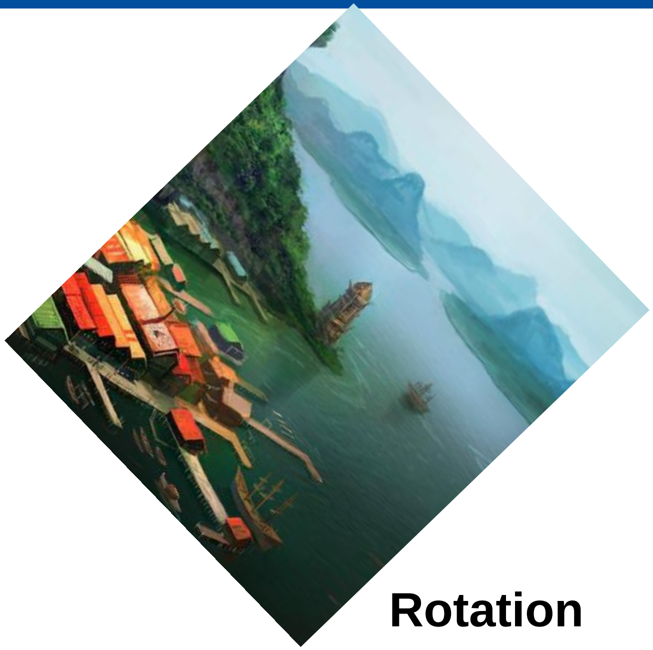
- Transformation:
  - maps points to other points and/or vectors to other vectors
- For images, shapes, *etc.*
  - A geometric transformation maps positions that define the object to other positions
  - Linear transformation means the transformation is defined by a linear function...



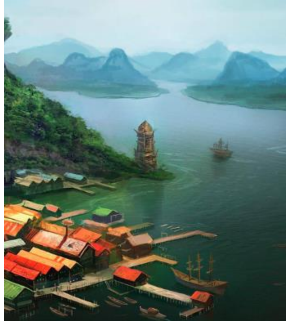
Some Examples



Original



Rotation



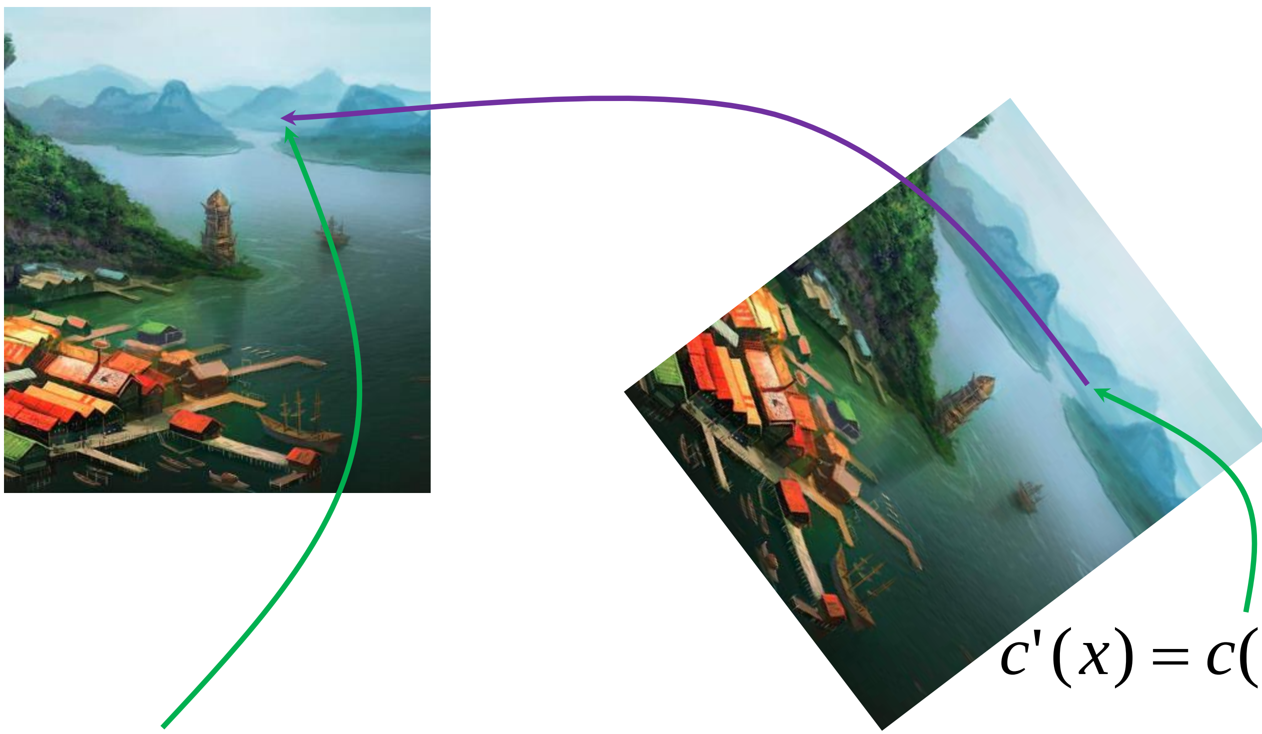
Uniform Scale

Shear

Nonuniform Scale

Mapping Function

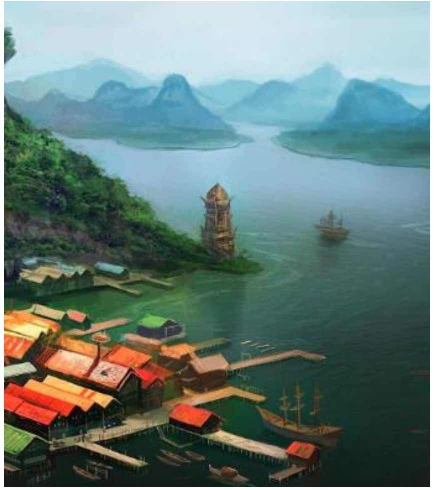
Mapping a position **x** in old image to a new position **x'**=**f**(**x**) in new image



$$c'(x) = c(f(x))$$

$$c[x] = [195,120,58]$$

Linear -vs- Nonlinear

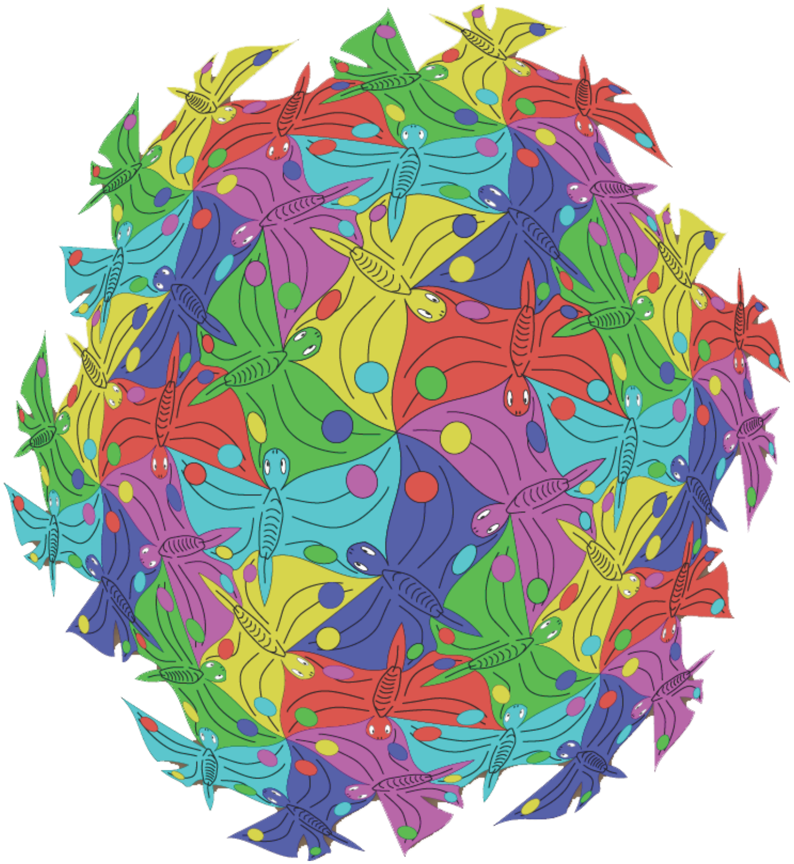


Nonlinear(swirl)

Linear(shear)

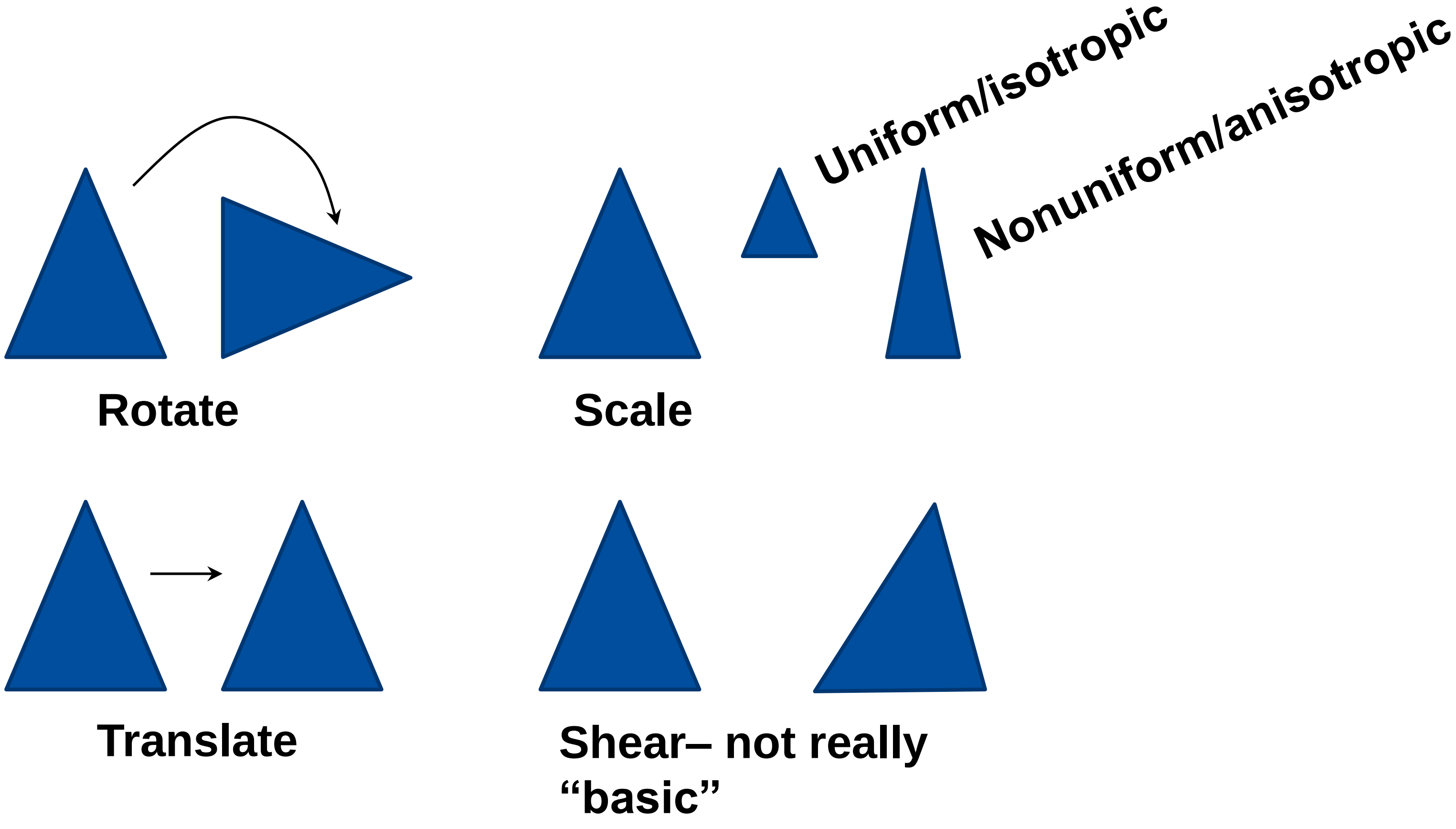
Instancing

- Reuse geometric descriptions
- Saves memory



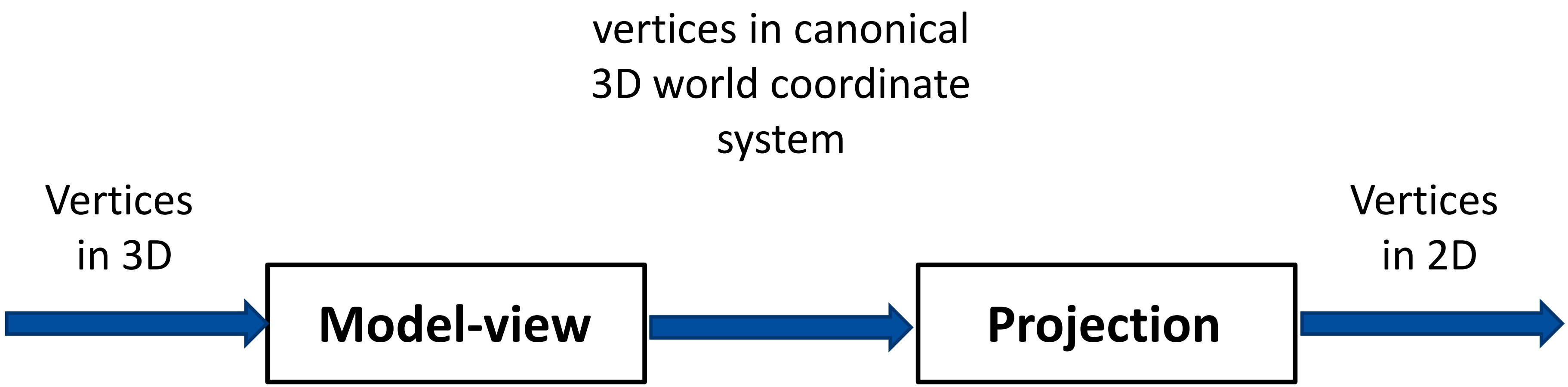
Basic Transformations

- Basic transforms are: rotate, scale, and translate
- Shear is a composite transformation!



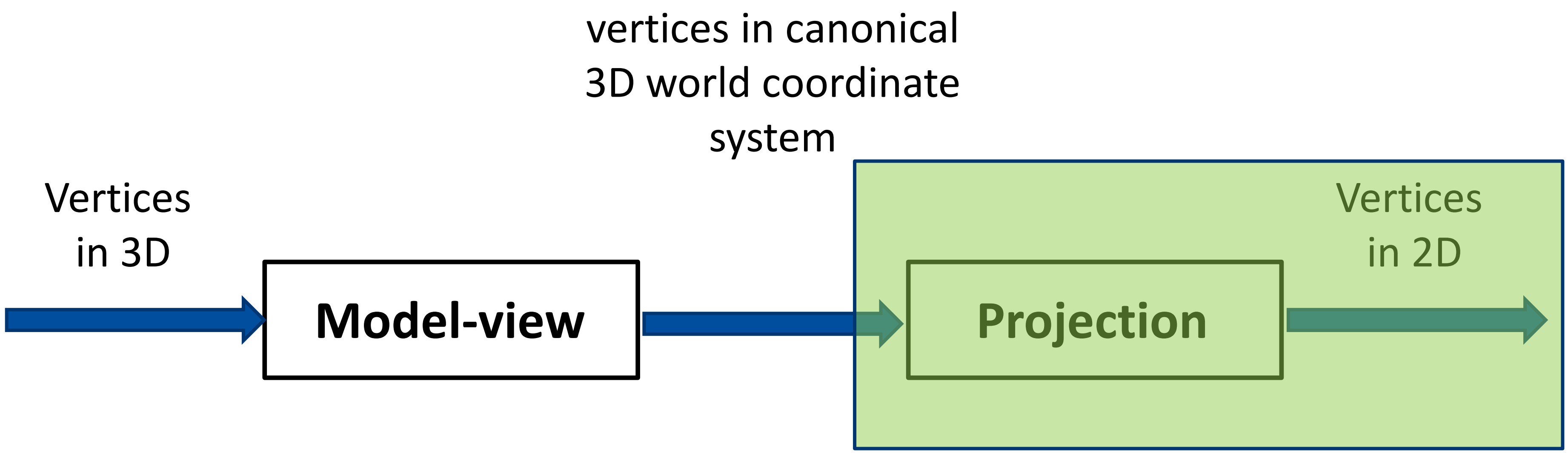
OpenGL Transformations Matrices

- Translate, rotate, scale objects
- Position the camera



OpenGL Transformation Matrices

- Projection from 3D to 2D



OpenGL Transformation Matrices

- Manipulated separately in OpenGL
  - (must set matrix mode) :

```
glMatrixMode (GL_MODELVIEW);
glMatrixMode (GL_PROJECTION);
```

Setting the Current Model-view Matrix

- Load or post-multiply

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity(); // very common usage
float m[16] = { … };
glLoadMatrixf(m); // rare, advanced
glMultMatrixf(m); // rare, advanced
```

- Use library functions

```
glTranslatef(dx, dy, dz);
glRotatef(angle, vx, vy, vz);
glScalef(sx, sy, sz);
```

Transformations

2D Transformations

Linear Functions in 2D

$$x' = f(x, y) = c_1 + c_2 x + c_3 y$$

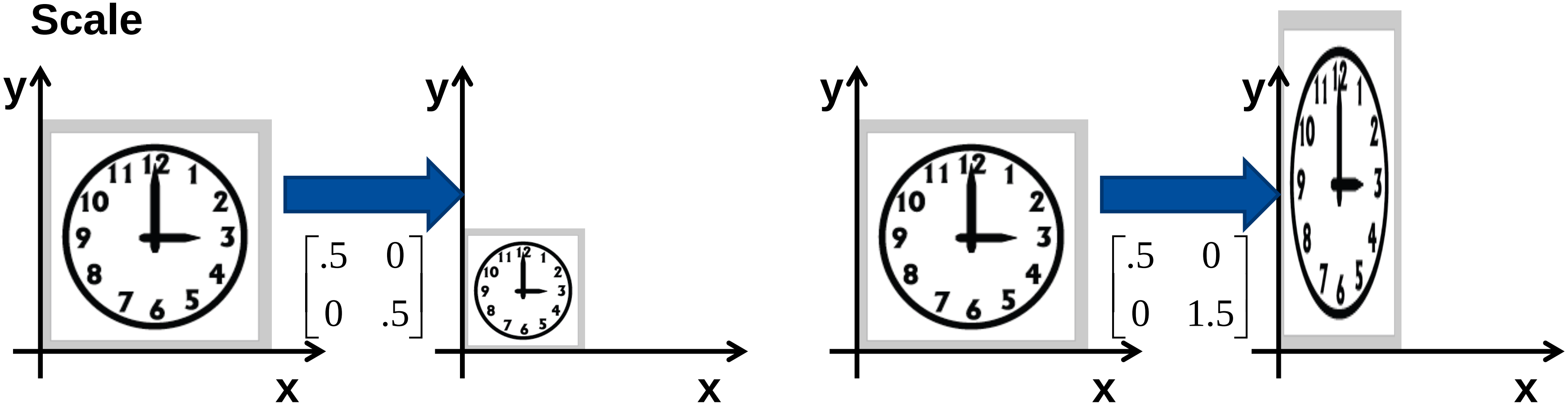
$$y' = f(x, y) = d_1 + d_2 x + d_3 y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} M_{xx} & M_{xy} \\ M_{yx} & M_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{t} + \mathbf{M} \cdot \mathbf{x}$$

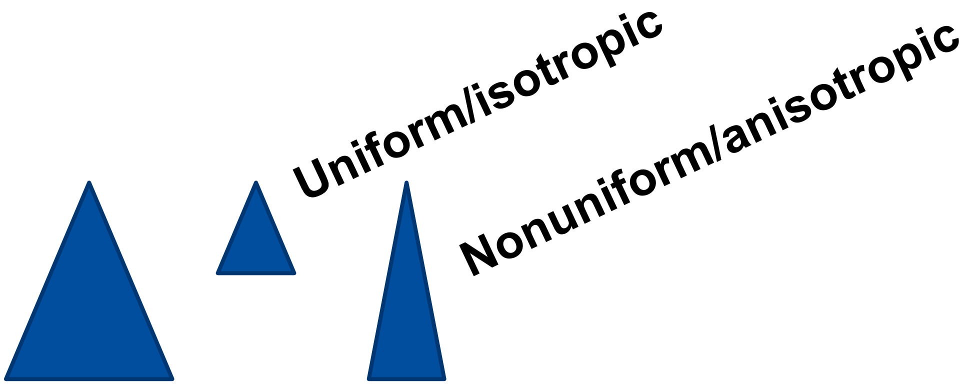
Scales

Scale



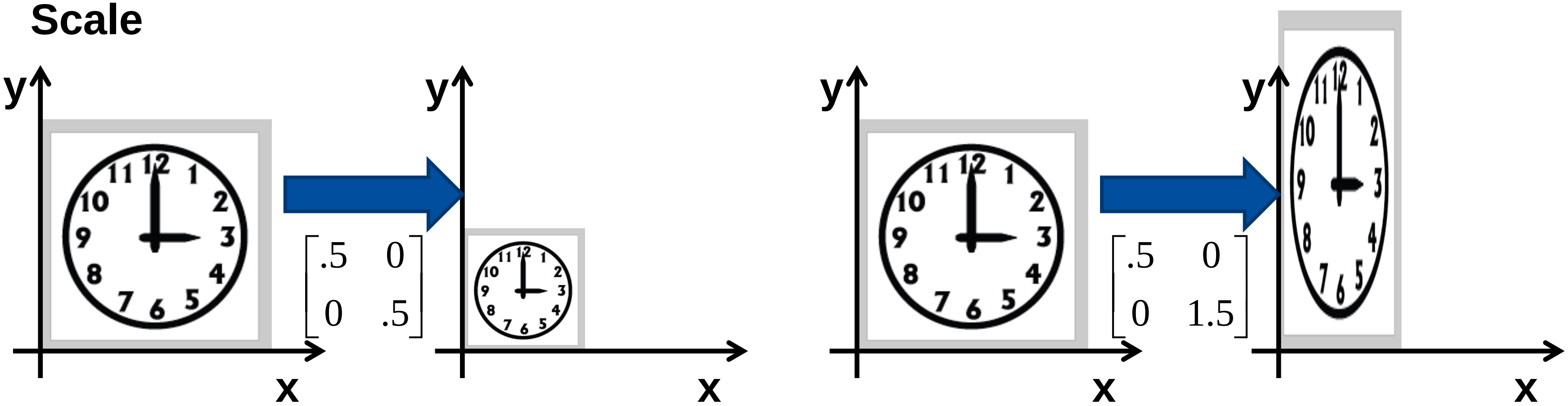
$$\begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix} \qquad \begin{bmatrix} .5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

Scales



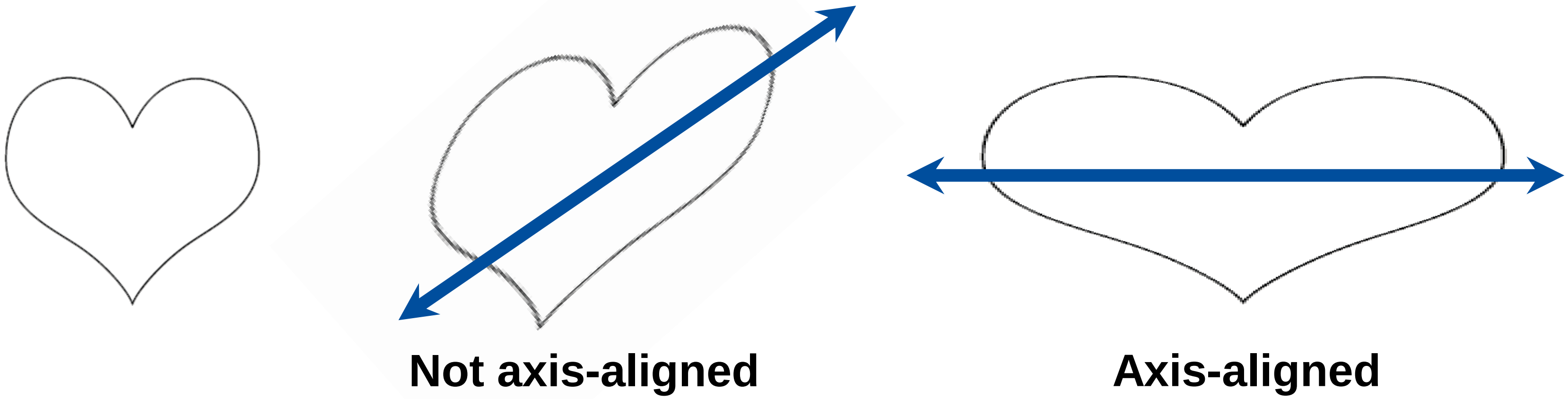
$$\mathbf{p}' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \mathbf{p}$$

Scale

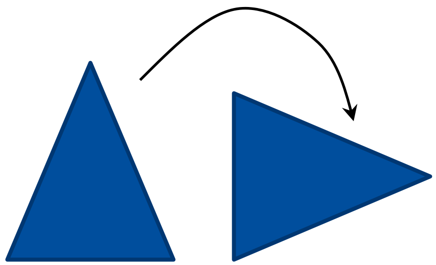


Scales

- Diagonal matrices
  - Diagonal parts are scale in X and scale in Y directions
  - Negative values flip
  - Two negatives make a positive (180 deg. rotation)
  - Axis-aligned scales



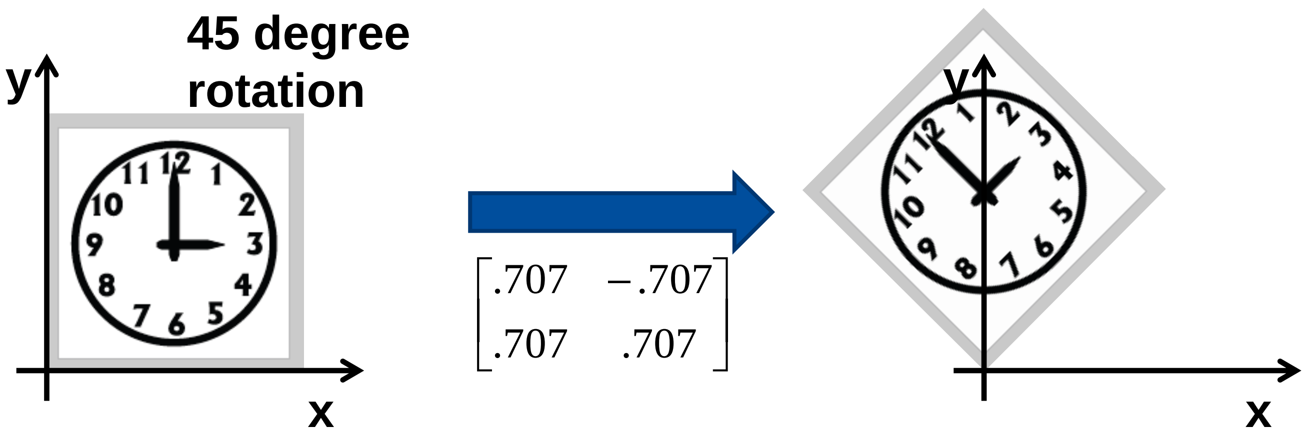
Rotations



Rotate

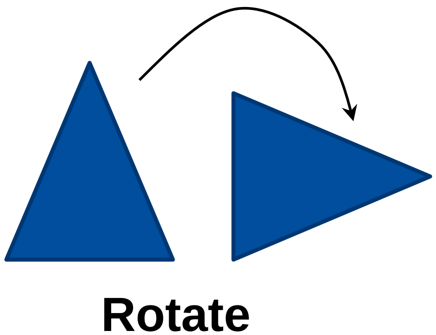
?

45 degree rotation

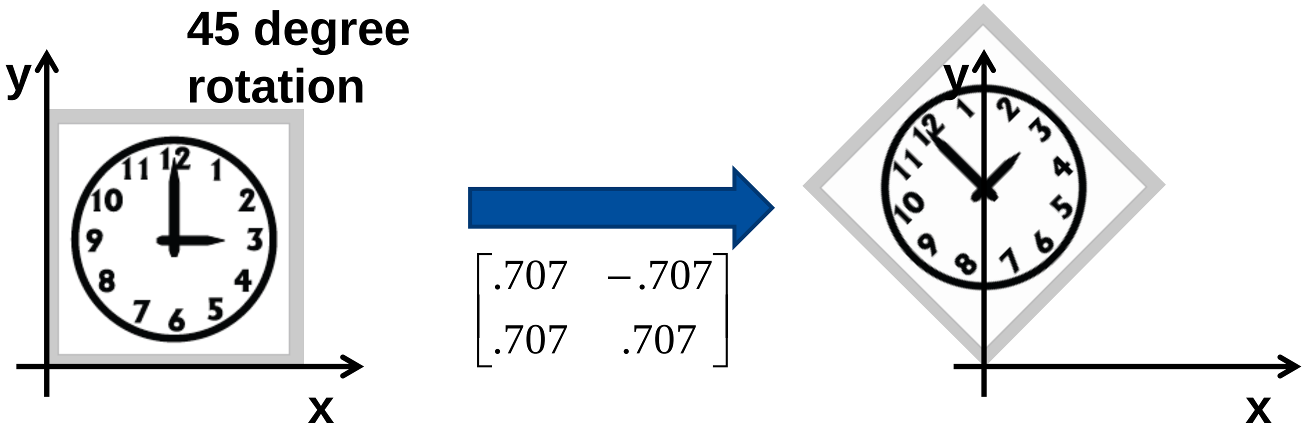


$$\begin{bmatrix} .707 & -.707 \\ .707 & .707 \end{bmatrix}$$

Rotations



$$\mathbf{p}' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \mathbf{p}$$



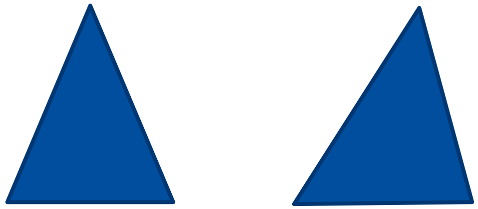
$$\begin{bmatrix} .707 & -.707 \\ .707 & .707 \end{bmatrix}$$

Rotations

- Rotations are counter-clockwise
- Consistent w/ right-hand rule
- Note:
  - Rotations of by zero degree give an identity
  - Rotations are modulo 360 (or 2π)

$$\mathbf{p}' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \mathbf{p}$$

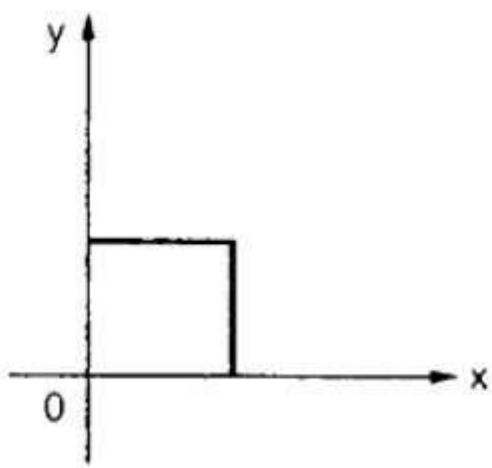
Shears



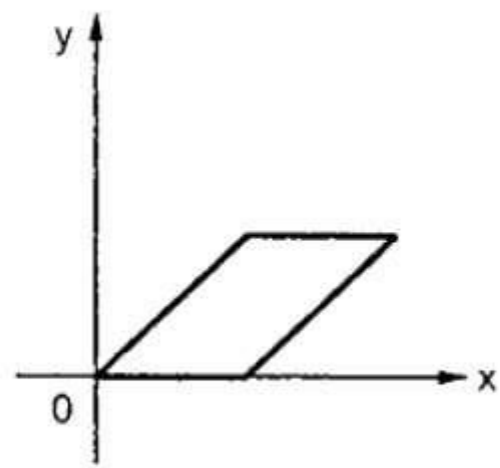
X-Shear

$$x' = x + H \cdot y$$

$$y' = y$$

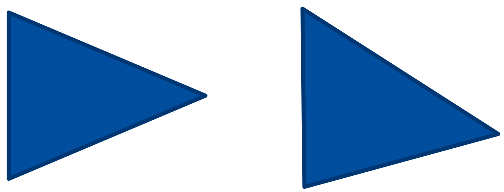


(a) Original object



(b) Object after x shear

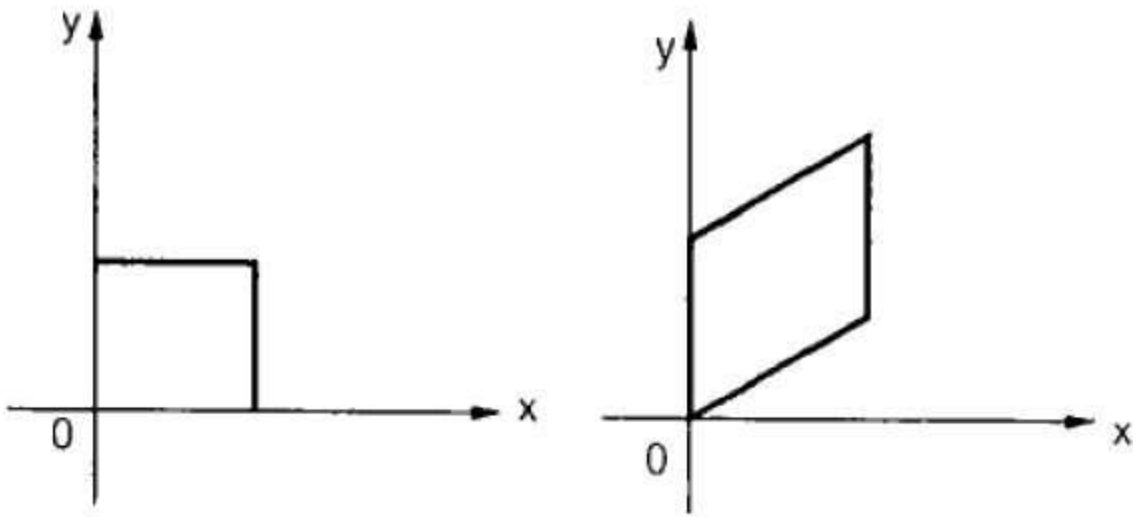
Shears (Y-shears)



Y-Shear

$$x' = x$$

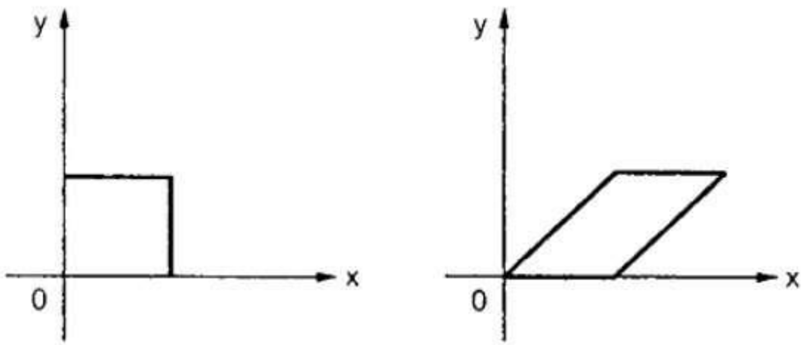
$$y' = y + h \cdot x$$



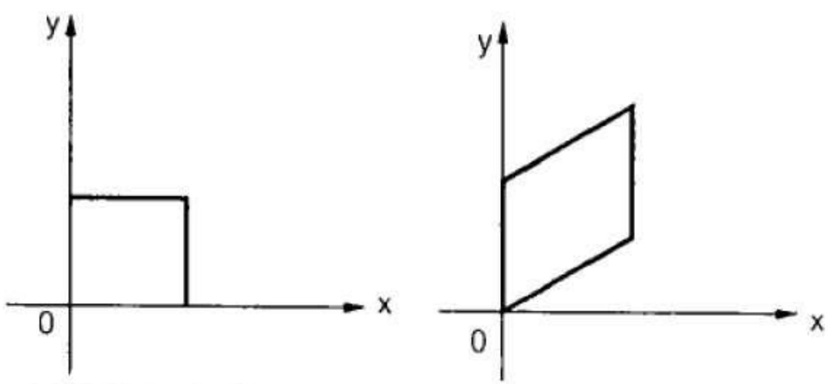
(a) Original object

(b) Object after y shear

Shears



(a) Original object (b) Object after x shear

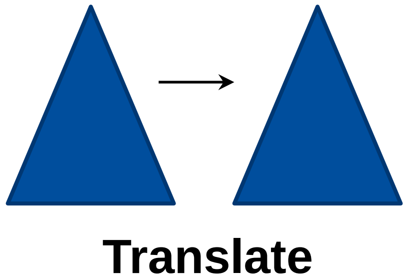


(a) Original object (b) Object after y shear

$$\mathbf{p}' = \begin{bmatrix} 1 & H_x \\ H_y & 1 \end{bmatrix} \mathbf{p}$$

Translation

- This is the not-so-useful way:



$$\mathbf{p}' = \mathbf{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Note that its not like the others.

Homogeneous Coordinates

- Move to one higher dimensional space
  - Append 1 at the end of the vectors

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \qquad \tilde{\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} w \cdot p_x \\ w \cdot p_y \\ w \end{bmatrix}, \quad w \neq 0$$

homogeneous coordinates

Homogeneous Translation

$$\mathbf{p}' = \mathbf{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\tilde{\mathbf{p}}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{p}}' = T\tilde{\mathbf{p}}$$

The tildes are for clarity to distinguish homogenized from non-homogenized vectors.

Homogeneous Others

- Scaling

$$\tilde{\mathbf{p}}' = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

- Rotation

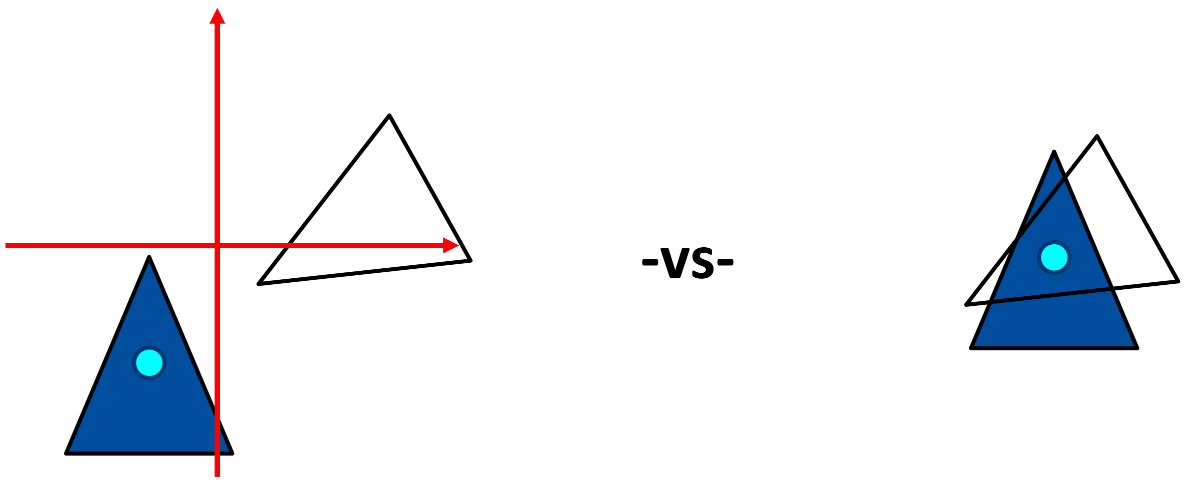
$$\tilde{\mathbf{p}}' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

- Shear

$$\tilde{\mathbf{p}}' = \begin{bmatrix} 1 & H_x & 0 \\ H_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{p}}$$

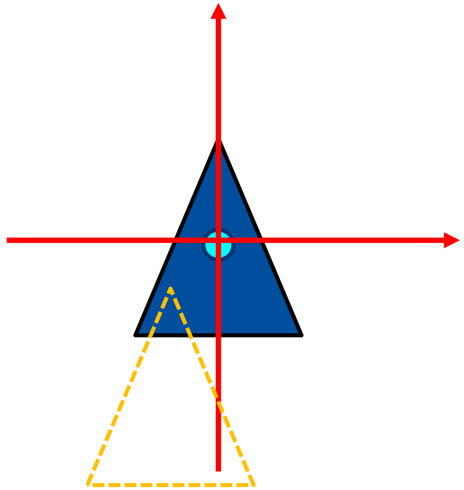
Compositing Matrices

- Rotations and scales always about the origin
- How to rotate/scale about another point?



Rotate about an arbitrary Point

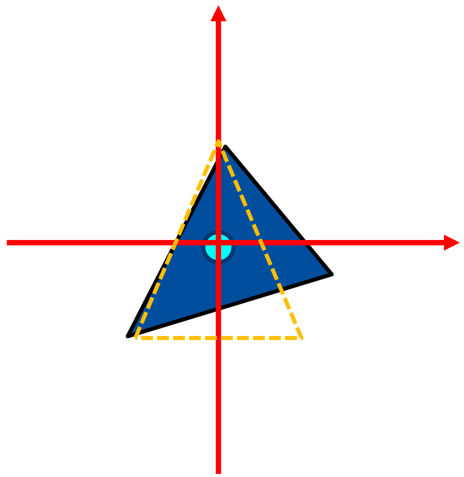
- Step 1: Translate point to origin



Translate(-C)

Rotate About Arb. Point

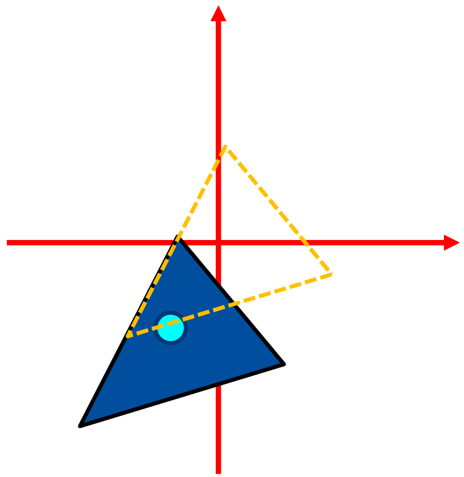
- Step 1: Translate point to origin
- Step 2: Rotate as desired



Translate(-C)
Rotate(θ)

Rotate About Arb. Point

- Step 1: Translate point to origin
- Step 2: Rotate as desired
- Step 3: Put back where it was



Translate(-C)
Rotate(θ)
Translate(C)

$$\tilde{p}' = TR(-T)\tilde{p} = A\tilde{p}$$

Order Matters!

- The order in which matrices appear matters

$$\mathbf{A} \cdot \mathbf{B} \neq \mathbf{BA}$$

- Some special cases work, but they are special
- But matrix multiplication is associative

$$(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})$$

- Think about efficiency when you have many points to transform...

Matrix Inverses

- In general: $\mathbf{A}^{-1}$ undoes effect of **A**
- Special cases:
  - Translation: negate t_x and t_y

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

  - Rotation: transpose (negate θ)

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

  - Scale: invert diagonal (axis-aligned scales)

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations

3D Transformations

3D Transformations

- Generally, the extension from 2D to 3D is straightforward
  - Vectors get longer by one
  - Matrices get one extra column and row
  - Scale, Translation, and Shear all basically the same
- Rotations get interesting

Translations

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D

Scales

$$\tilde{\mathbf{A}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

$$\tilde{\mathbf{A}} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D (Axis-aligned!)

Shears

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & 0 \\ h_{yx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

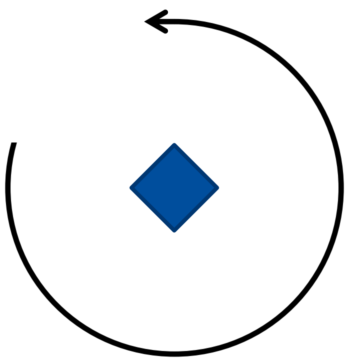
$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D **(Axis-aligned!)**

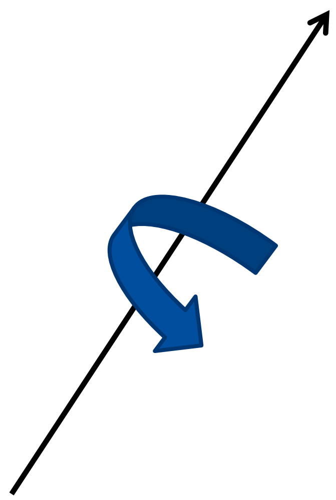
Rotations

- 3D Rotations fundamentally more complex than in 2D
  - 2D: angle of rotation
  - 3D: angle and axis of rotation

2D



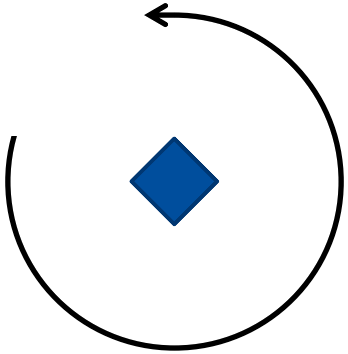
-VS-

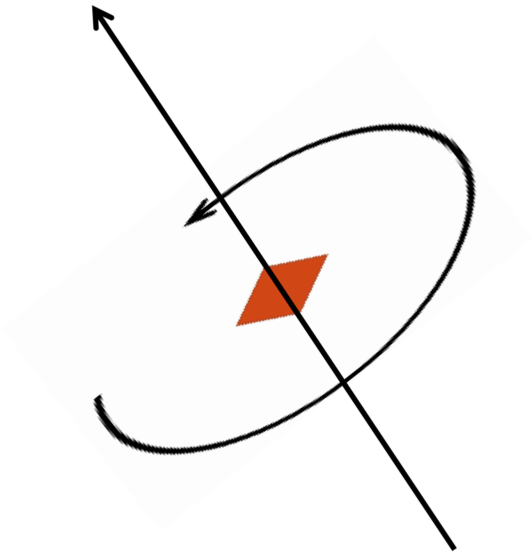


3D

Axis-aligned 3D Rotations

- 2D rotations implicitly rotate about a third out of plane axis



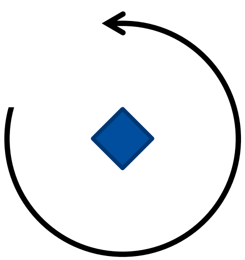


Axis-aligned 3D Rotations

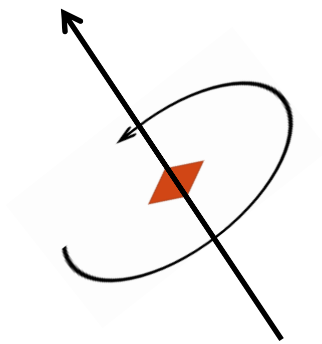
- 2D rotations implicitly rotate about a third out of plane axis

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

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



Note: looks same as $\tilde{\mathbf{R}}$



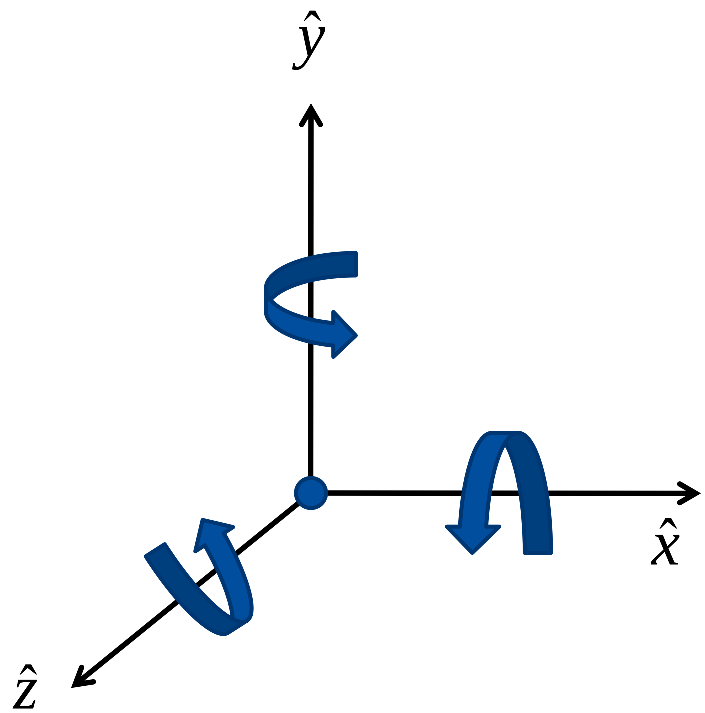
Axis-aligned 3D Rotations

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

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

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

"Z is in your face"



Axis-aligned 3D Rotations

- Also known as "direction-cosine" matrices

$$\mathbf{R}_{\hat{x}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \mathbf{R}_{\hat{y}} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{z}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Radial Axis

- Can be built from axis-aligned matrices:

$$\mathbf{R} = \mathbf{R}_{\hat{z}} \cdot \mathbf{R}_{\hat{y}} \cdot \mathbf{R}_{\hat{x}}$$

- Euler Angles

- In OpenGL
  - void glRotated(angle, x, y, z)