February 27, 2025, Brno

Author: David Procházka

Projections & Transformations

Graphic Application Development

MENDELU
Faculty
of Business
and Economics

Table of content

1. Projections
2. Orthographic projection
3. Perspective projection
4. Setting the view matrix
5. Projection Examples
6. Transformations
7. Composition of transformations
8. Summary

Rendering a scene

(Camera analogy)

1. Camera – setting the camera into the scene – view transformation.
2. Object positioning – insert object into the scene – model transformation.
3. Lenses – selecting a lenses – projection transformation.
4. Shot – mapping the final image into the window – cropping etc..

Matrices updates

- Model and view matrices (modelview matrix) is continuously changed
- It must be usually send to shader in the rendering method.
- Projection is usually same the whole time, except:
  1. We need to change the appearance of the scene (CAD).
  2. We changed the shape of our window.

Projection & transformation settings

We do not use any projection/transformation command directly, but we use similar commands that generate for us matrices that are provided to the shaders. They are mentioned only for explanation of the principles.

Table of content

1. Projections
2. Orthographic projection
3. Perspective projection
4. Setting the view matrix
5. Projection Examples
6. Transformations
7. Composition of transformations
8. Summary

Orthographic projection

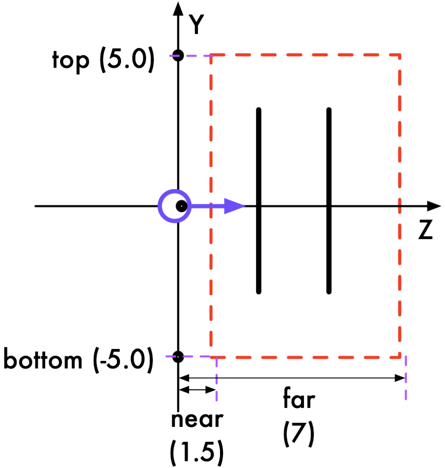
- Orthographic projection is give by a rectangular prism.
- Distance from an object does not influence it's size.
- Used mostly in CAD applications etc.

Setting the orthographic projection

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)
```

- `near` – **distance** to the near plane,
- `far` – **distance** to the far plane,
- other values are ranges on the particular axes.

Orthographic projection – side view



Mathematical background

Projection is represented by 4×4 matrix. This matrix is applied on vertices.

$$\begin{bmatrix} \frac{2}{right - left} & 0 & 0 & \frac{-(right+left)}{right-left} \\ 0 & \frac{2}{top-btm} & 0 & \frac{-(top+btm)}{top-btm} \\ 0 & 0 & \frac{-2}{far-near} & \frac{-(far+near)}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Details: `http://learnwebgl.brown37.net/08_projections/projections_ortho.html`

Table of content

1. Projections
2. Orthographic projection
3. Perspective projection
4. Setting the view matrix
5. Projection Examples
6. Transformations
7. Composition of transformations
8. Summary

Perspective projection

- The scene is given by a frustrum (pyramid without top).
- Therefore, the nearer the object is, the larger is it's appearance. (I takes larger share of the needle slice in the particular distance.)
- There are, two well-known commands:

Setting a perspective projection

```
void glFrustum(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top, GLdouble near,
GLdouble far)
```

Setting a perspective projection (more user friendly)

```
void gluPerspective(GLdouble fovy, GLdouble aspect,
GLdouble near, GLdouble far)
```

Perspective projection – side view

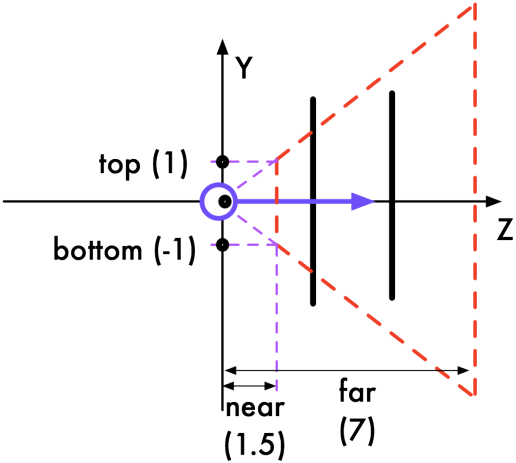


Table of content

1 Projections
2 Orthographic projection
3 Perspective projection
4 Setting the view matrix
5 Projection Examples
6 Transformations
7 Composition of transformations
8 Summary

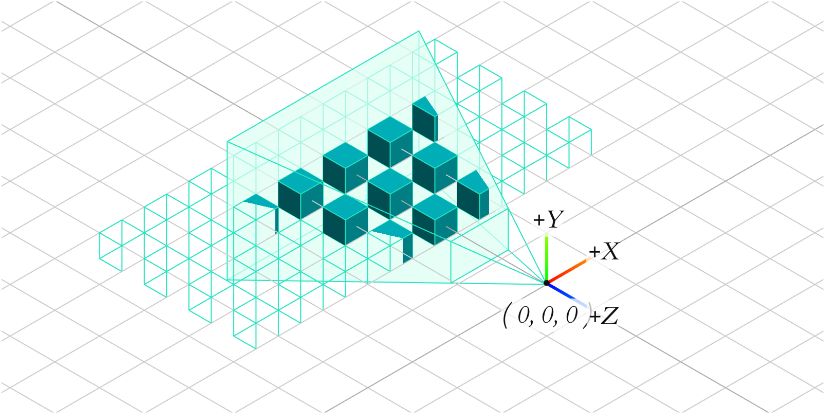
View matrix

Setting an observer

```
gluLookAt(
GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz,
GLdouble upx, GLdouble upy, GLdouble upz
)
```

- `GLdouble eyex, GLdouble eyey, GLdouble eyez` – camera position,
- `GLdouble centerx, GLdouble centery, GLdouble centerz` – a point we look at,
- `GLdouble upx, GLdouble upy, GLdouble upz` – up vector.

Principle of the observer



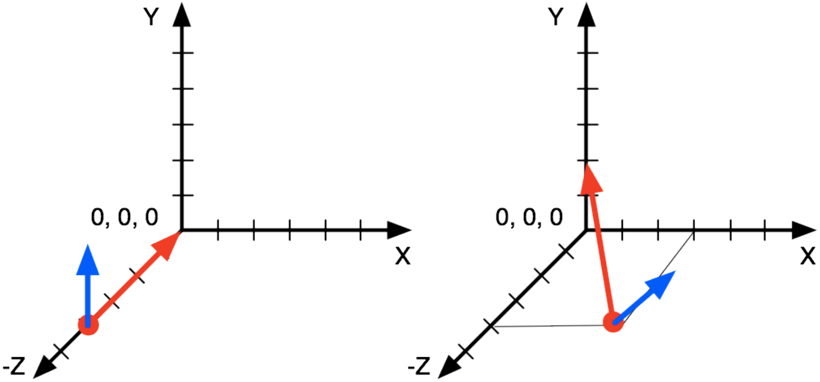
Source: `https://jsantell.com/model-view-projection/`

Common observer setting

- Move the observe out of the scene alongside the *z* axis.
- Up vector is along the *y* axis.

```
gluLookAt(
  0.0, 0.0,-5.0,
  0.0, 0.0, 0.0,
  0.0, 1.0, 0.0)
```

Examples of observer setting



Common ”step back” and complex orientation.

How we can do it?

Just generate appropriate matrices using embedded functions. E.g.:

```
/// lets create a matrix
QMatrix4x4 projectionMatrix;
/// make it identity matrix
projectionMatrix.setToIdentity();
/// fill it with projection matrix
projectionMatrix.ortho(-2.0, 2.0, -2.0, 2.0, 0.0, 100.0);
/// send it to the shader
m_program->setUniformValue(m_matrixUniform, projectionMatrix);
```

And do not forget on depth testing!

Table of content

1. Projections
2. Orthographic projection
3. Perspective projection
4. Setting the view matrix
5. Projection Examples
6. Transformations
7. Composition of transformations
8. Summary

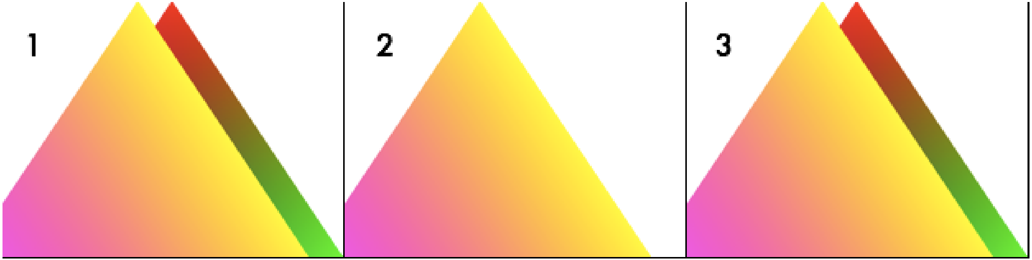
Examples of the orthographic projection

Let us have two triangles give by following positions and colors:

```
GLint triangle[] = {
     0,  5, 5,
    -5, -5, 5,
     5, -5, 5,
     1,  5, 2,
    -4, -5, 2,
     6, -5, 2};
```

```
GLfloat colors[] = {
    1.0, 0.0, 0.0,
    0.0, 1.0, 0.0,
    0.0, 0.0, 1.0,
    1.0, 1.0, 0.0,
    1.0, 1.0, 0.0,
    1.0, 0.0, 1.0};
```

Three results



Their settings

1. Common situation: `glOrtho(-5.0,5.0,-5.0,5.0,0.0,5.0);`
   `gluLookAt(0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0);`
2. Triangle in the back is missing:
   `glOrtho(-5.0,5.0,-5.0,5.0,0.0,4.0);`
   `gluLookAt(0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0);`
3. We changed the observer point.
   `glOrtho(-5.0,5.0,-5.0,5.0,0.0,5.0);`
   `gluLookAt(0.0,0.0,0.0,0.0,0.0,10.0,0.0,1.0,0.0);`

Three results (2)

Their settings (2)

1. Observer moved alonside the *z* axis:
```
glOrtho(-5.0,5.0,-5.0,5.0,0.0,5.0);
gluLookAt(0.0, 0.0,-1.0,0.0,0.0,10.0,0.0,1.0,0.0)
```
2. Scale of the axes is doubled:
```
glOrtho(-10.0,10.0,-10.0,10.0,0.0,5.0);
gluLookAt(0.0,0.0,0.0,0.0,0.0,10.0,0.0,1.0,0.0);
```
3. Up vector is changed:
```
glOrtho(-10.0,10.0,-10.0,10.0,0.0,5.0);
gluLookAt(0.0,0.0,0.0,0.0,0.0,10.0,1.0,1.0,0.0);
```

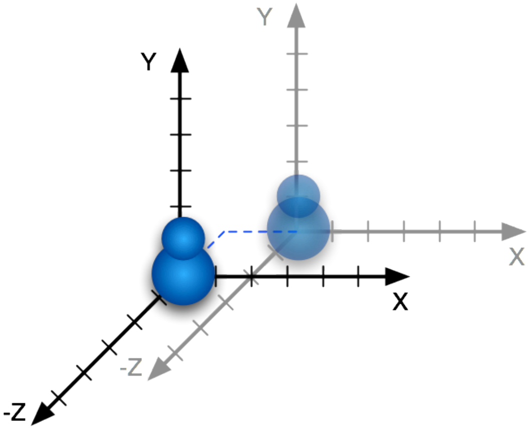
Table of content

1. Projections
2. Orthographic projection
3. Perspective projection
4. Setting the view matrix
5. Projection Examples
6. Transformations
7. Composition of transformations
8. Summary

Object transformations

- There are three general geometric transformations:
    1. rotation,
    2. translation,
    3. scale.
- All complex movements are given by composition of these elementary transformations.
- All transformations are represented as 4×4 matrices that are applied on vertex coordinates.

Object translation



Translation of an object along the *x* and *z* axis.

Object translation

glTranslate*

For object translation, we use command
```
void glTranslated(GLdouble x, GLdouble y, GLdouble z)
void glTranslatef(GLfloat x, GLfloat y, GLfloat z).
```

Values x, y and z are the offsets along respective axes.

Mathematical background

The command generates T matrix, where t_x, t_y and t_z are values from the command. This matrix is multiplied by previous transformation matrix or a vertex coordinates.

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T^{-1} is inverse transformation matrix.

Calculation

Let us translate vertex X with coordinates [10, 10, 10][1] by 10 units along the x axis. Let X' is the new position of the vertex.

$$X' = T.X = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1x + 0y + 0z + t_x w \\ 0x + 1y + 0z + t_y w \\ 0x + 0y + 1z + t_z w \\ 0x + 0y + 0z + 1w \end{bmatrix}$$

$$= \begin{bmatrix} x + t_x w \\ y + t_y w \\ z + t_z w \\ w \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 10 + 10 \\ 10 + 0 \\ 10 + 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

[1] The w value is always 1 because of the matrix multiplication.

Inverse transformation

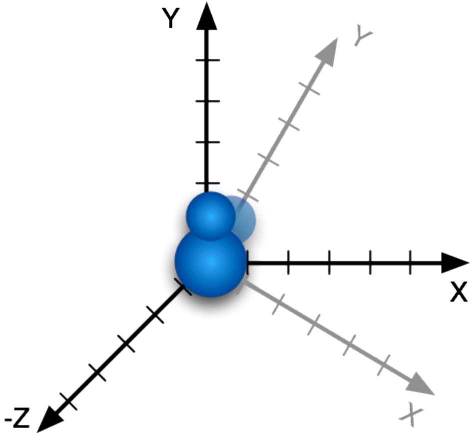
Let us move the X' vertex back using the inverse transf. matrix T^{-1}:

$$X = T^{-1}.X' =$$

$$\begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1x + 0y + 0z + -t_x w \\ 0x + 1y + 0z + -t_y w \\ 0x + 0y + 1z + -t_z w \\ 0x + 0y + 0z + 1w \end{bmatrix}$$

$$= \begin{bmatrix} x' - t_x w \\ y' - t_y w \\ z' - t_z w \\ w' \end{bmatrix} = \begin{bmatrix} x' - t_x \\ y' - t_y \\ z' - t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 20 - 10 \\ 10 - 0 \\ 10 - 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

Object rotation



Rotation of an object around the z axis by 40° clockwise.

Object rotation

glRotate*

For rotation around given axis clockwise, we use command

```
void glRotated(GLdouble angle,
GLdouble x, GLdouble y, GLdouble z)
void glRotatef(GLfloat angle,
GLfloat x, GLfloat y, GLfloat z).
```

The first parameter is rotation angle, the rest of the parameters define around which axis or axes the rotation is made. The default orientation of the rotation is counterclockwise.

Mathematical background

$$\texttt{glRotate*(a, 1, 0, 0)}: \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(a) & -sin(a) & 0 \\ 0 & sin(a) & cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\texttt{glRotate*(a, 0, 1, 0)}: \begin{bmatrix} cos(a) & 0 & sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(a) & 0 & cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\texttt{glRotate*(a, 0, 0, 1)}: \begin{bmatrix} cos(a) & -sin(a) & 0 & 0 \\ sin(a) & cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse tranf. matrix is is same, we just use $-a$ instead of a.

Calculation of the rotation

Let us have X vertex with coordinates [10, 20, 30], which will be rotated by 45° around the z axis. The X' is the new position of the vertex.

$$X' = R.X = \begin{bmatrix} cos(45^\circ) & -sin(45^\circ) & 0 & 0 \\ sin(45^\circ) & cos(45^\circ) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 20 \\ 30 \\ 1 \end{bmatrix} =$$

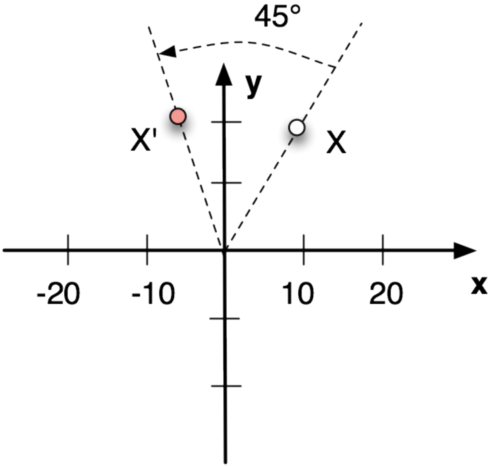
$$= \begin{bmatrix} 10.cos(45^\circ) + -20.sin(45^\circ) + 0.30 + 0.1 \\ 10.sin(45^\circ) + 20.cos(45^\circ) + 0.30 + 0.1 \\ 10.0 + 20.0 + 30.1 + 0.1 \\ 10.0 + 20.0 + 30.0 + 1.1 \end{bmatrix} = \begin{bmatrix} -7^2 \\ 21 \\ 30^3 \\ 1 \end{bmatrix}$$

---

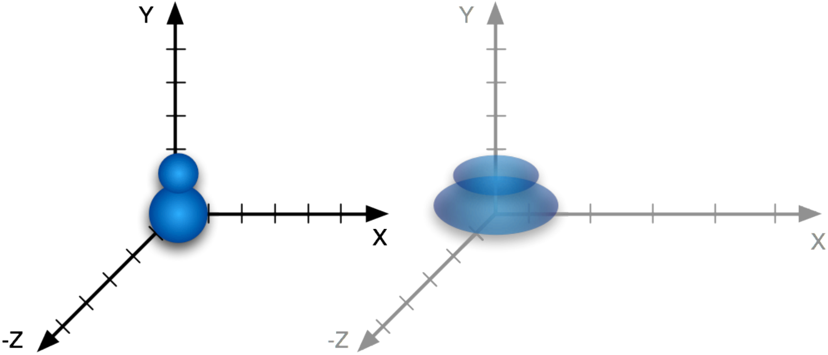
[2] rounded result

[3] The z axis and the w coordinate are not changed

Sketch of the result



Scale of an object



Doubling the scale of an object in the *x* direction.

Scale of an object

`glScale*`

We can scale an object using the commmands:

```
void glScaled( GLdouble x, GLdouble y, GLdouble z)
void glScalef(GLfloat x, GLfloat y, GLfloat z).
```

The parameters are scale coefficients in respective directions.

Mathematical background

The command generated S matrix, where s_x, s_y and s_z are scales from the command. We will apply the matrix on the vertex coordinated as in the previous cases.

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale computation

Let us change the scale of the X vertex with coordinates [10, 10, 10] in a following way: $s_x = 2$, $s_y = 2$ a $s_z = 1$. Let X' is the transformed vertex.

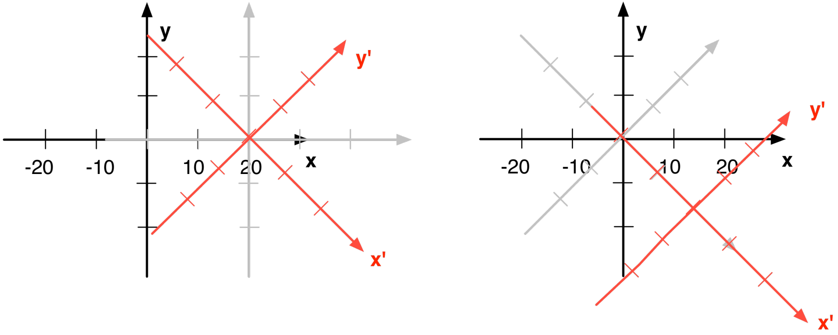
$$X' = S.X = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix} =$$

$$= \begin{bmatrix} 210 \\ 210 \\ 110 \\ 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 10 \\ 1 \end{bmatrix}$$

Table of content

1 Projections
2 Orthographic projection
3 Perspective projection
4 Setting the view matrix
5 Projection Examples
6 Transformations
7 Composition of transformations
8 Summary

Composition of transformations



Left: translate (20, 0, 0) and rotation (45, 0, 0, 1)
Right: rotation (45, 0, 0, 1) and translation (20, 0, 0)

Beware of the order!

Matrix multiplication is not commutative

It is utmost important to take into account the order of the transformations. The transformations are represented as matrices and matrix multiplication is not commutative!

Table of content

1 Projections
2 Orthographic projection
3 Perspective projection
4 Setting the view matrix
5 Projection Examples
6 Transformations
7 Composition of transformations
8 Summary

Takeaway

- What is the difference between orthogonal and perspective projection.
- How to set an observer into the scene.
- What is the mathematical principle behind projections and transformations.
- What are the basic transformations and how to create complex motions using their composition.