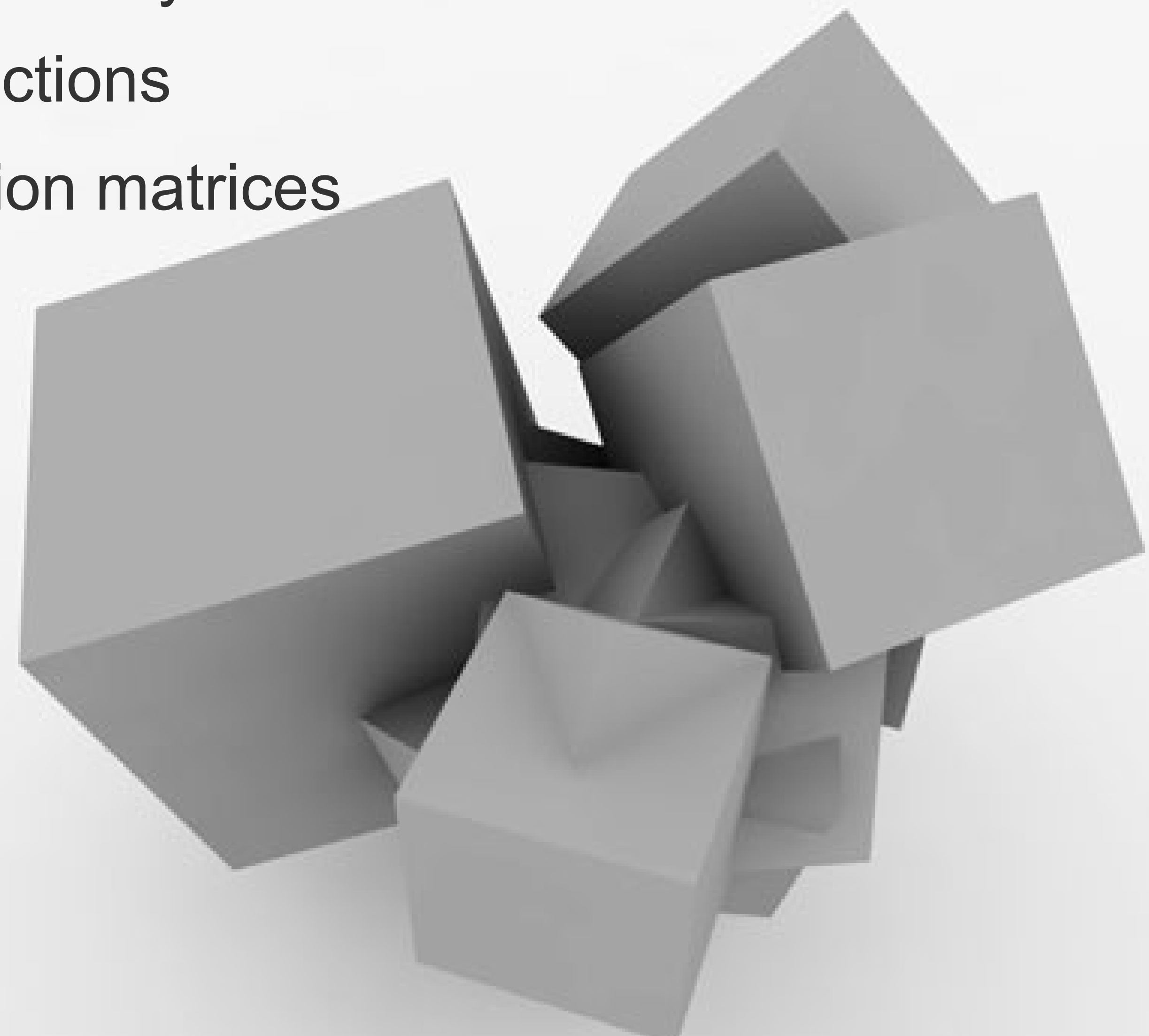
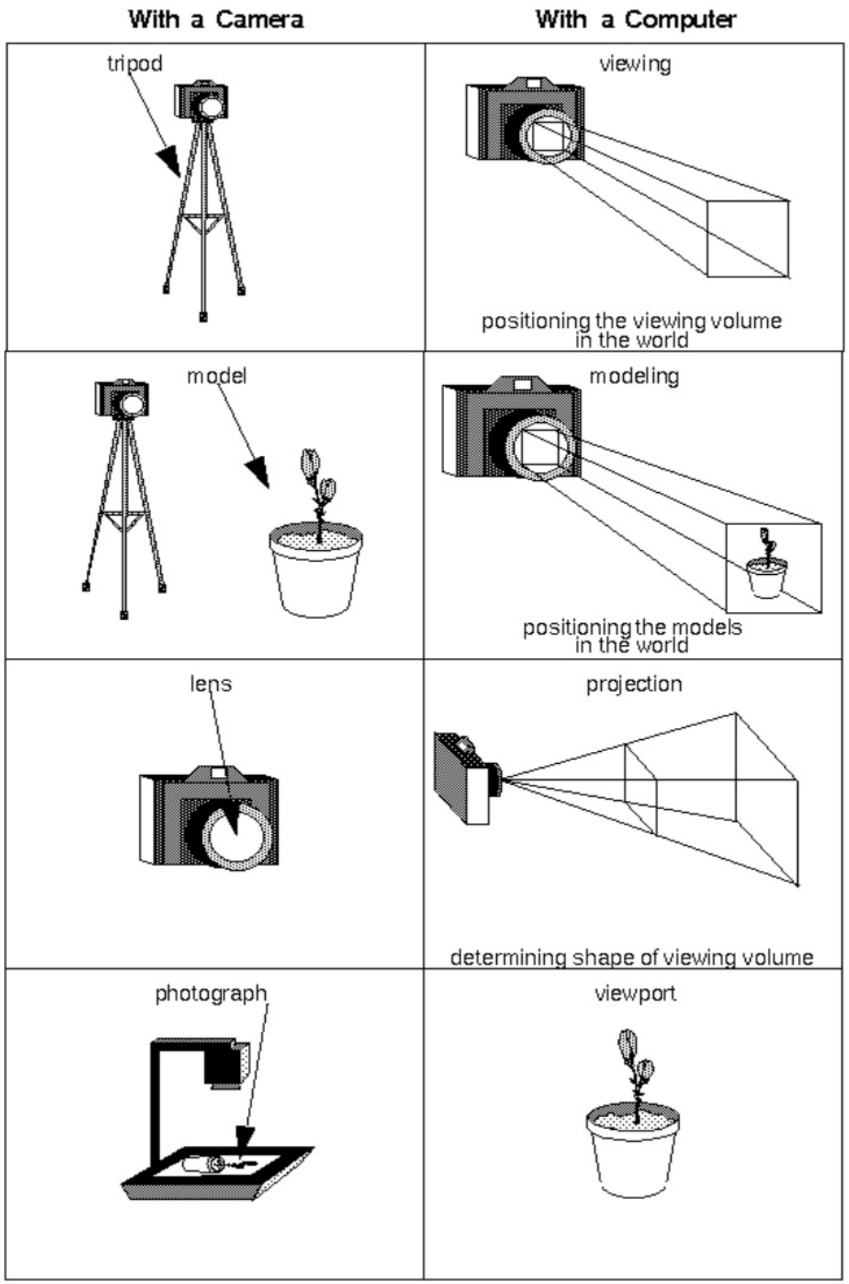
Transformations, Viewing and Projections

- Coordinate systems
- glm functions
- Projection matrices



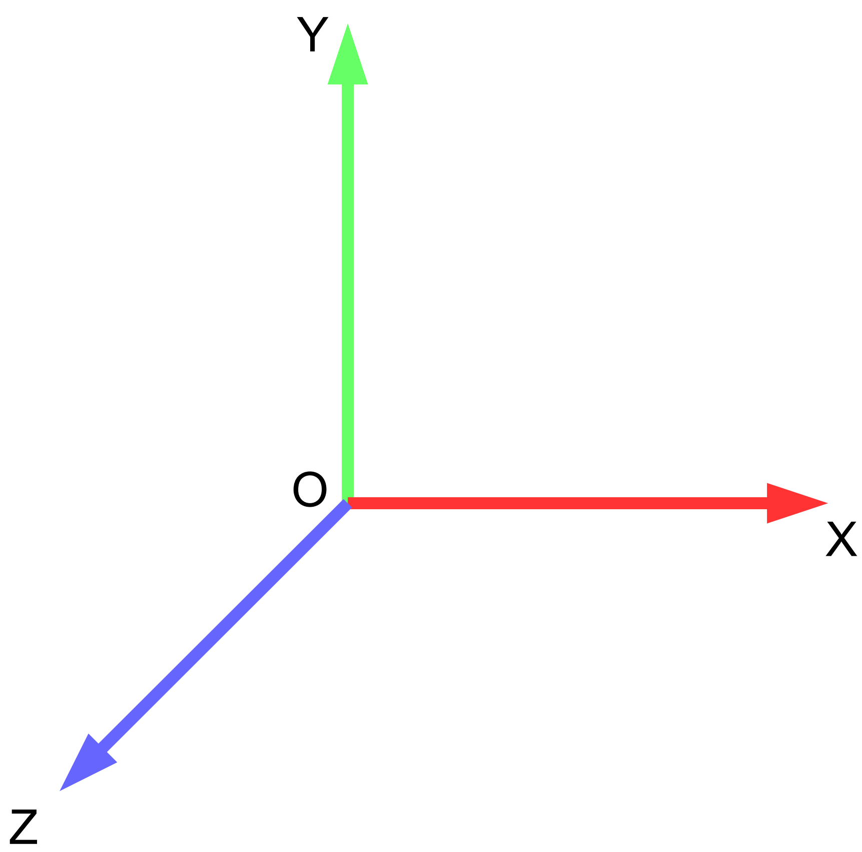


In OpenGL we will use 4 x 4 matrices to control:

- Object position, orientation, scale
- Camera position and orientation
- Camera projection propoerties
- and more...

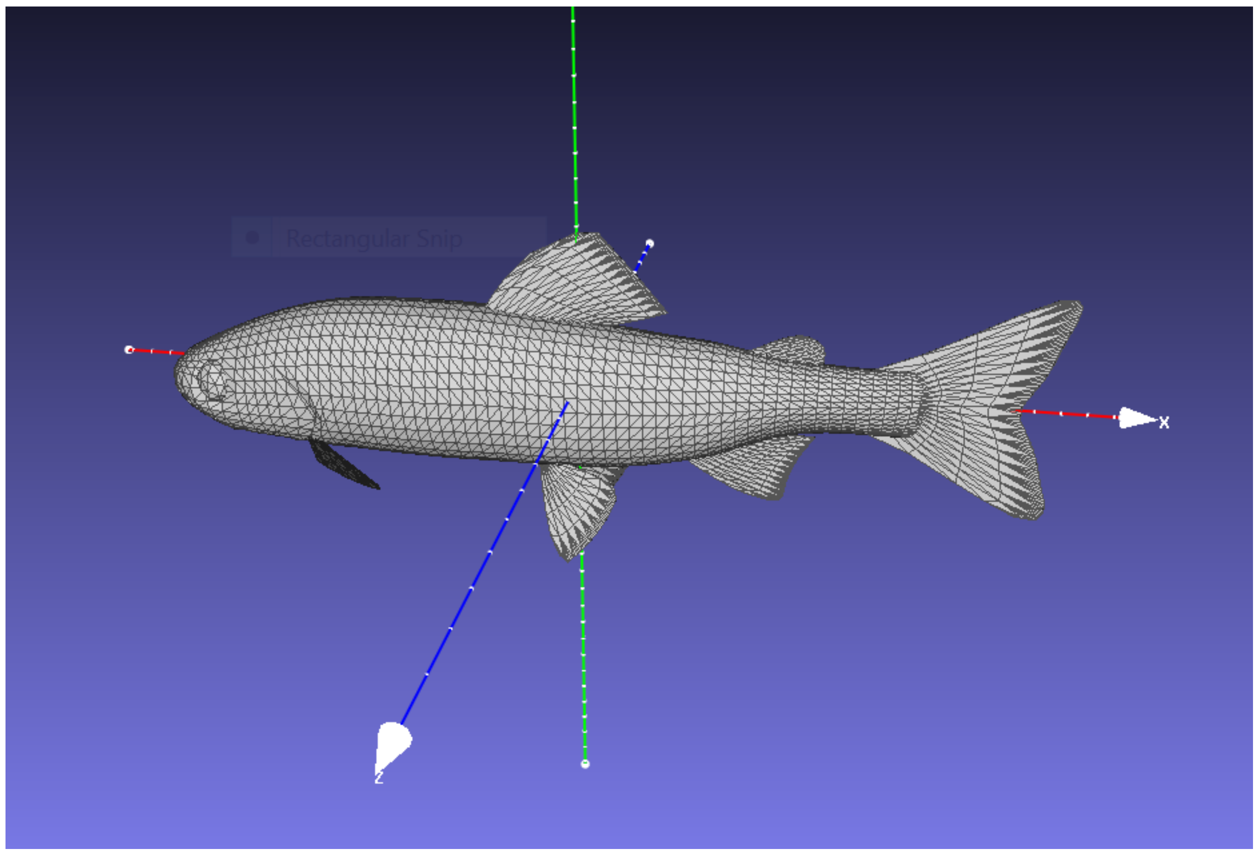
Coordinate systems

- Also called “coordinate frame”
  - In 3D : 3 basis vectors and origin point

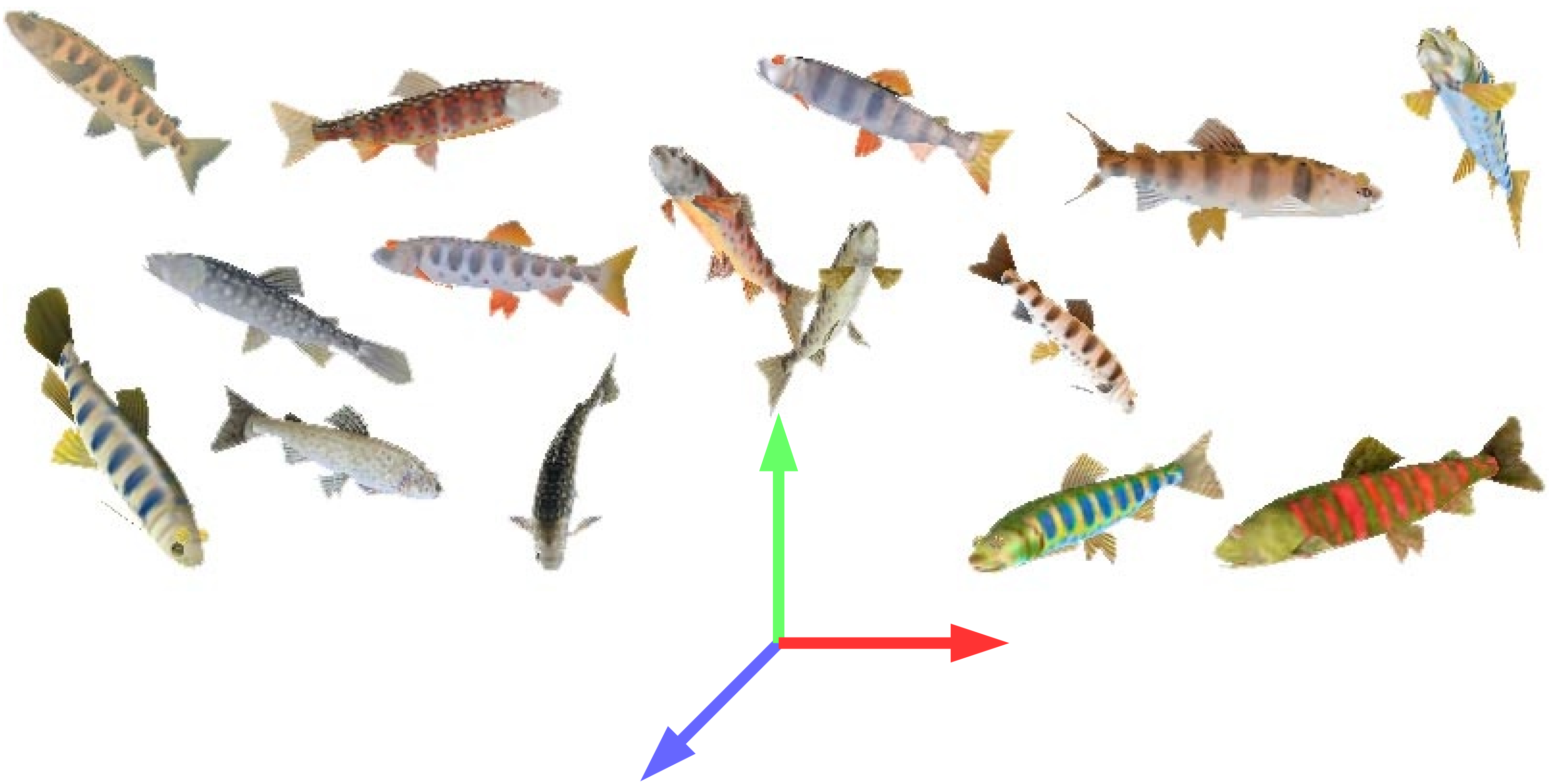


Object coordinates

- Also “object space”



World coordinates



Camera coordinates

- “camera space” or “eye space”

Matrix Names

- Common terminology for referring to matrices
  - Modeling matrix, **M** : place objects in world
  - Viewing matrix, **V** : camera position and orientation
  - Projection matrix, **P** : camera viewing volume shape

- As transformations between frames
  - **M** transforms from <u>object coordinates</u> to <u>world coordinates</u>
  - **V** transforms from <u>world coordinates</u> to <u>camera coordinates</u>
  - **P** transforms from <u>camera coordinates</u> to <u>clip coordinates</u>

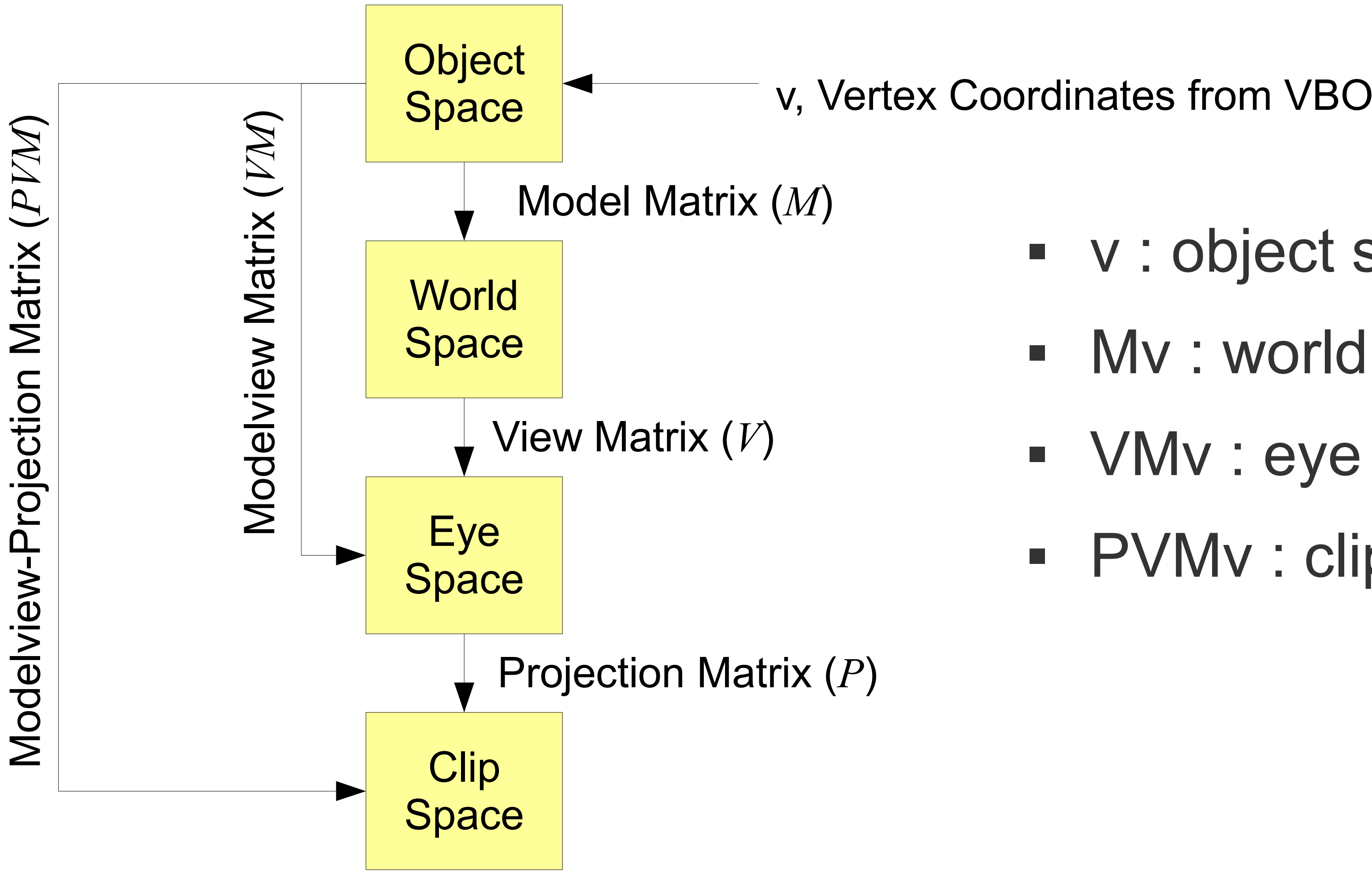
Matrix Names

- The matrix product **VM** is sometimes referred to a the “model-view matrix”
- The matrix product **PVM** … “model-view-projection” matrix

- As transformations between frames
  - **VM** transforms from object coordinates to camera coordinates
  - **PVM** transforms from object coordinates to clip coordinates

Transformations

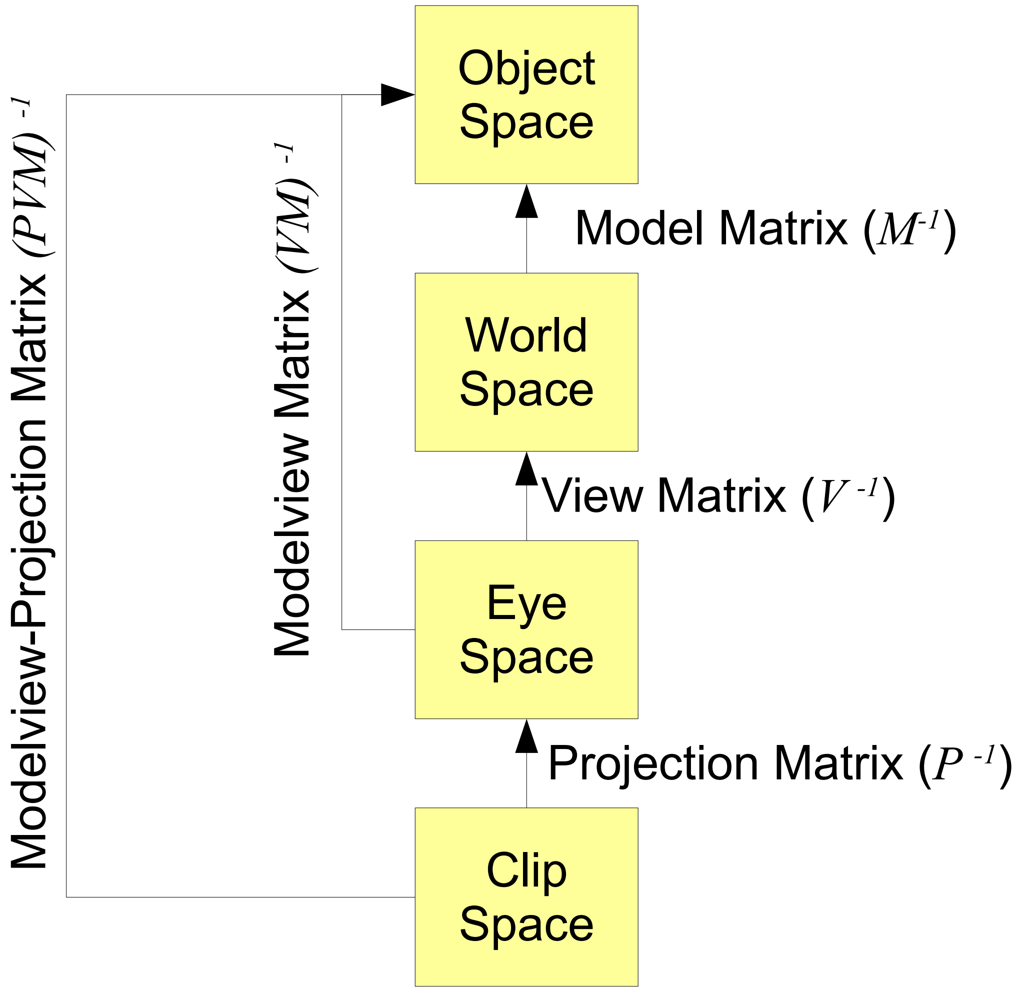
Matrix multiplication transforms points, vectors from one space to another.



- v : object space
- Mv : world space
- VMv : eye space
- PVMv : clip space

Transformations

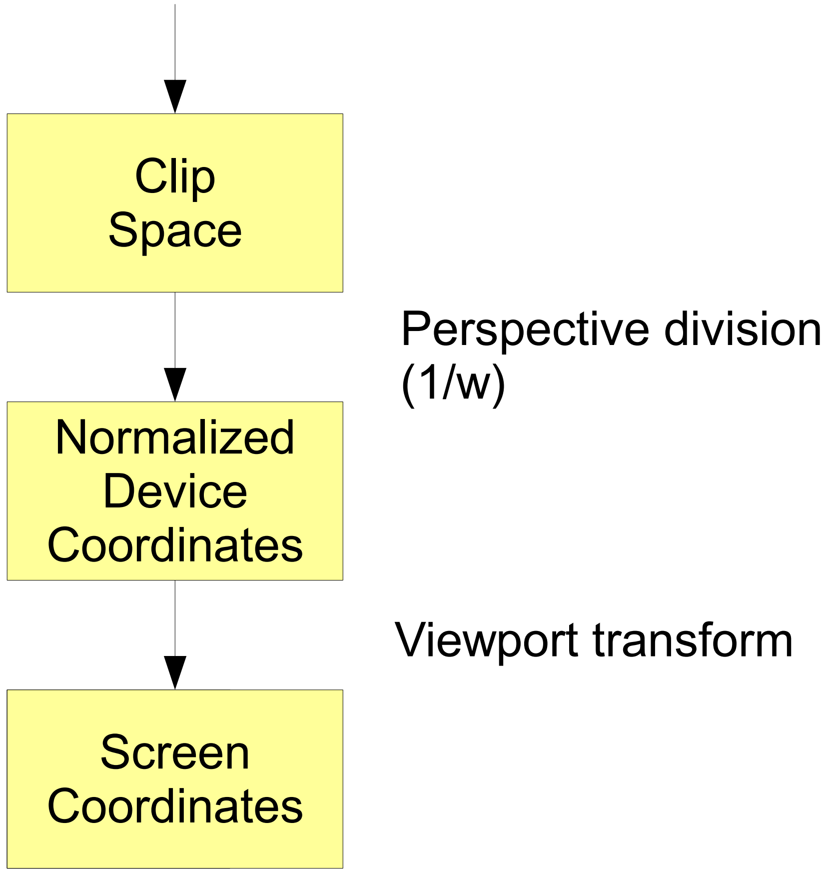
Matrix multiplication transforms points, vectors from one space to another.



- if p is in eye space
  - $V^{-1}p$: world space
  - $M^{-1}V^{-1}p$: object space
  - $(VM)^{-1}p$: object space

Clip space

- We are working with homogeneous coordinates
  - 3d points, vectors are represented as 4-component vectors



- OpenGL expects gl_Position to be given in clip coordinates
- Normalized device coordinates range from -1 to +1 in each direction
  - Things outside this range are clipped
- OpenGL performs “perspective division” and “viewport transformation” internally

glm Intro

- For more info see https://glm.g-truc.net/
- Header only – no compilation, linking required
- Types are named the same as glsl types
  - vec2, vec3, vec4
  - mat2, mat3, mat4
- Operators overloaded as in glsl
  - mat4 M;
  - vec4 p1, p2;
  - p2 = M*p1;
- Functions named the same as in glsl
  - dot, cross, reflect,

glm Modeling Transformations

- In glm/gtc/matrix_transform.hpp
  - mat4 translate(vec3 t);
    - t : translation vector
  - mat4 T = glm::translate(glm::vec3(x, y ,z));

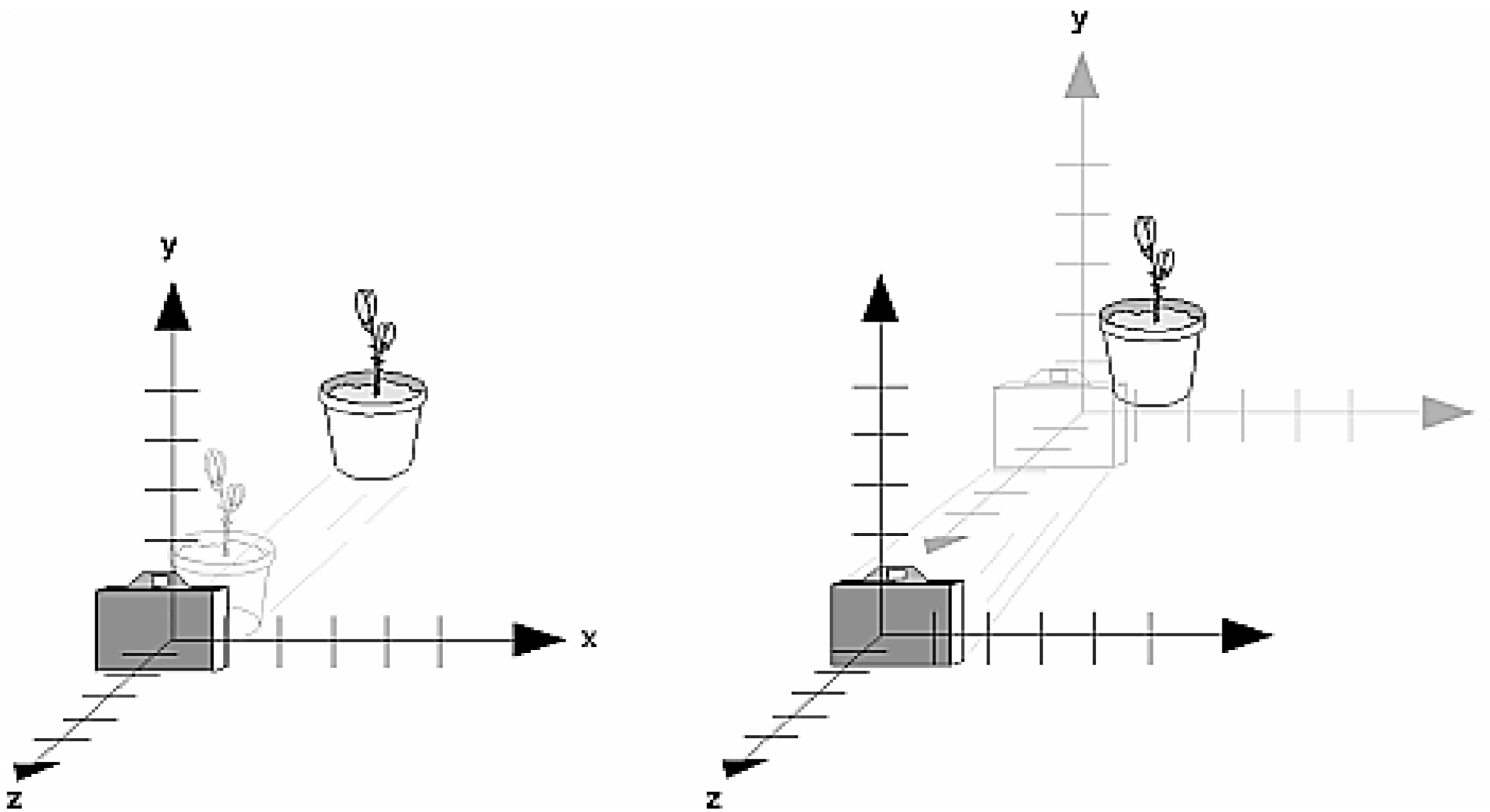
Resulting matrix, T, translates vertices by (x,y,z)

glm Modeling Transformations

- mat4 scale(vec3 s);
  - s : scale factors
- mat4 rotate(float angle, vec3 axis);
  - angle : rotation angle in degrees
    - in radians if GLM_FORCE_RADIANS is declared
  - axis : rotation axis
- Combine (by multiplying) several matrices to get more complex transformations
  - `glm::mat4 M = R*S;`
  - `glm::mat4 VM = V*M;`
  - `glm::mat4 PVM = P*V*M;`

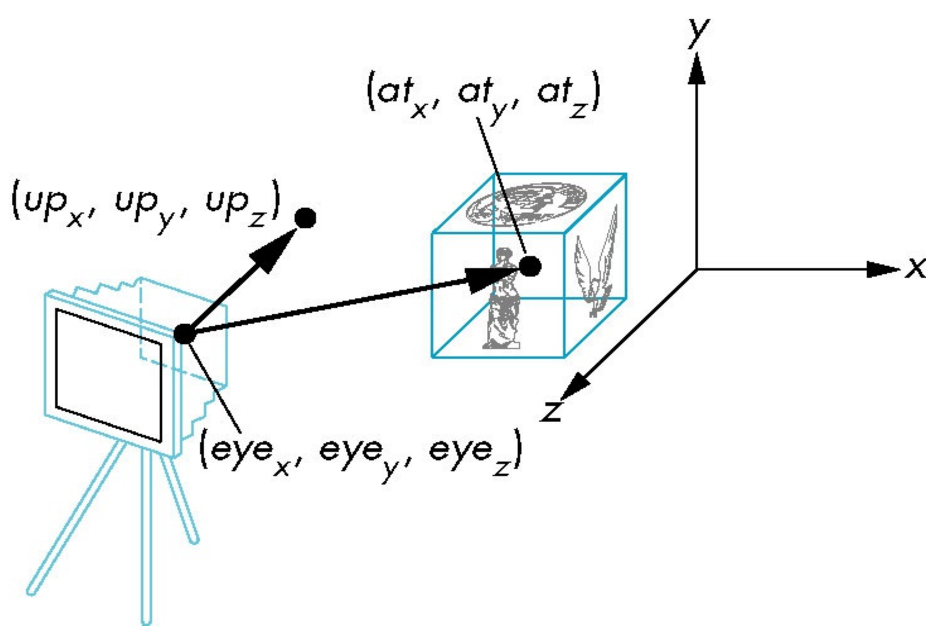
Positioning the camera

- Two equivalent transformations:
  - Apply transformation M to every object in the scene
  - Apply transformation M^{-1} to the camera frame

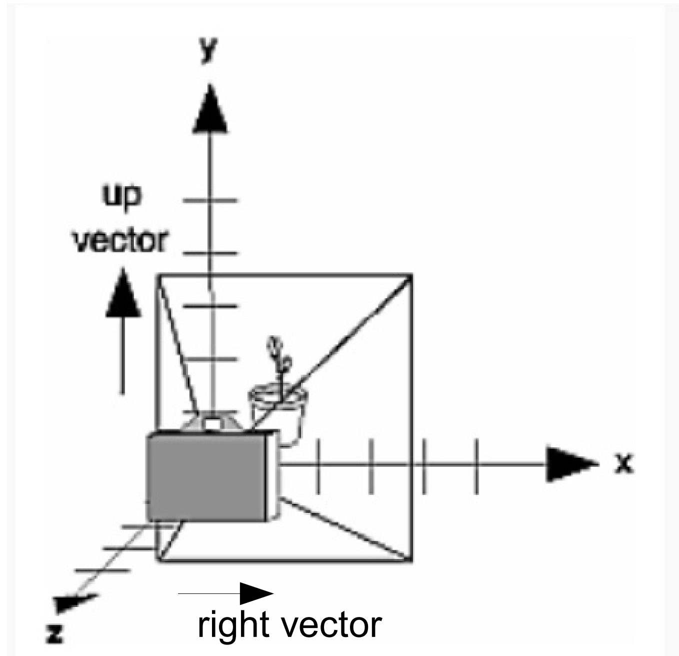


glm Viewing Transformations

- Construct the view matrix, V
- mat4 lookAt(vec3 eye, vec3 at, vec3 up);
  - eye : camera position
  - at : point the camera is aimed at
  - up : vertical direction in final image



Camera frame



- Look vector : the direction the camera is pointing
- Up vector : from camera position toward top of viewing volume
- Right vector : from camera position toward right side of viewing volume

The view matrix

The view matrix transforms points/vectors from world coordinates to camera coordinates. When the view matrix consists of only rotations and translations, the matrix has a special interpretation:

$$\mathbf{V} = \begin{bmatrix} r_x & r_y & r_z & v_{14} \\ u_x & u_y & u_z & v_{24} \\ -l_x & -l_y & -l_z & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where **r**=right, **u**=up and **l**=look vector in world coordinates.

- This is a direct consequence of the fact that $\mathbf{V}^{\mathrm{T}}$ converts the camera frame into the world frame

Getting look, right, up

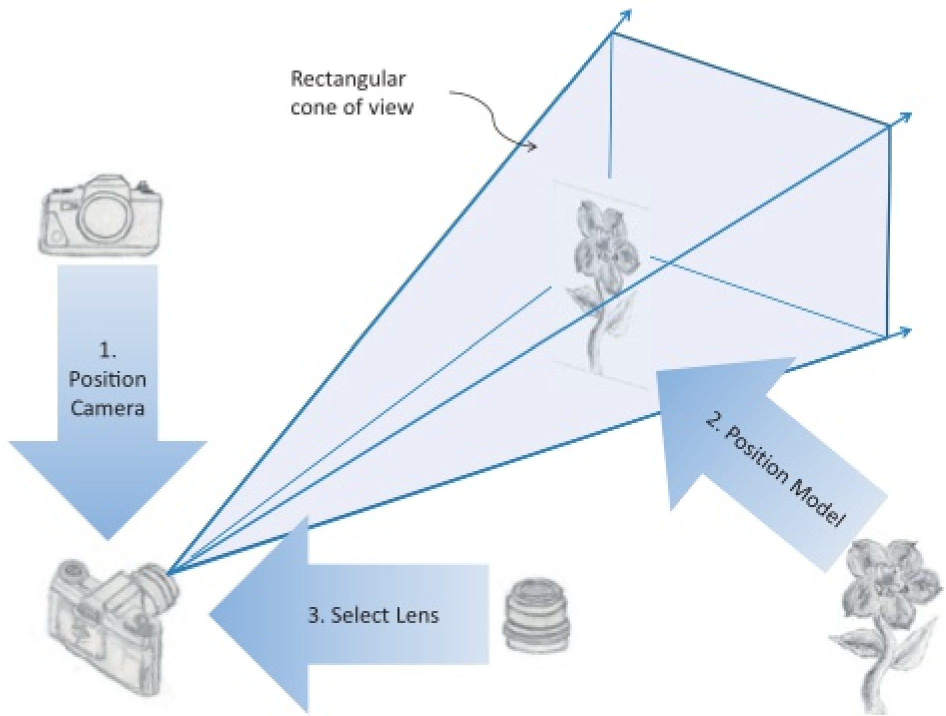
```
//extract camera look, right and up vectors from view matrix
//note: glm access to matrices is [col][row]
glm::vec3 look(-V[0][2], -V[1][2], -V[2][2]);
glm::vec3 up(V[0][1], V[1][1], V[2][1]);
glm::vec3 right(V[0][0], V[1][0], V[2][0]);
```

Why would you want these vectors?

First-person camera control:

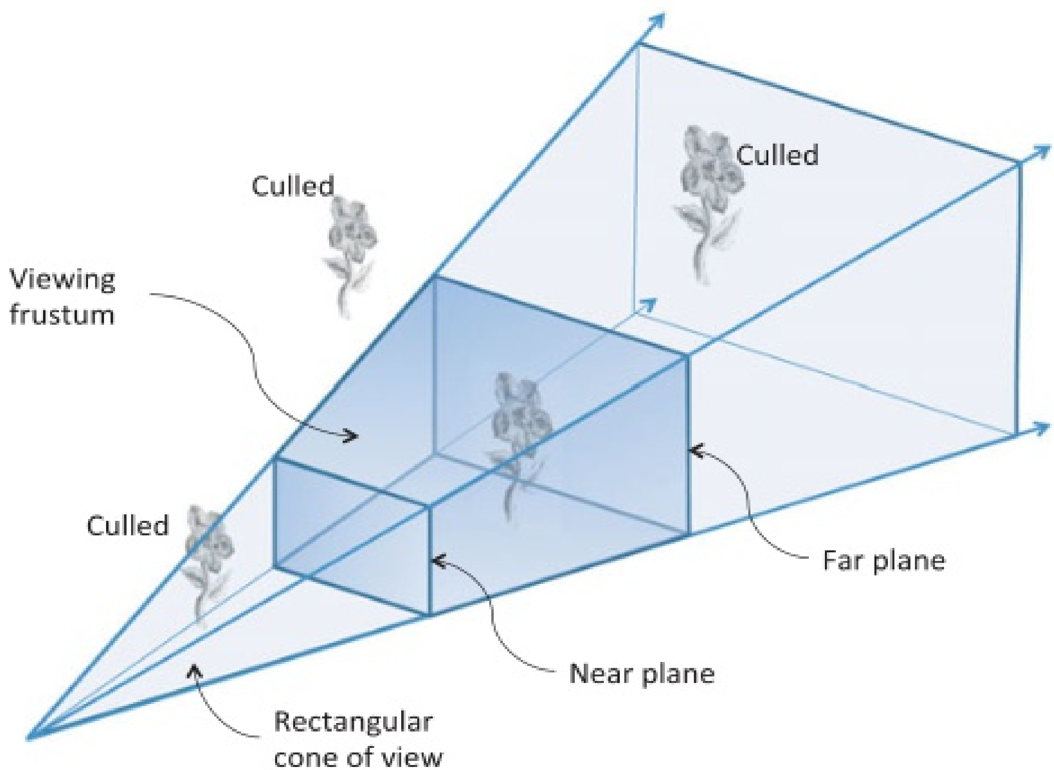
- Move forward : translate in look direction.
- Look left/right : rotate about up vector.
- Look up/down : rotate about right vector.
- Roll or tilt view : rotate about look vector.

Projection matrices



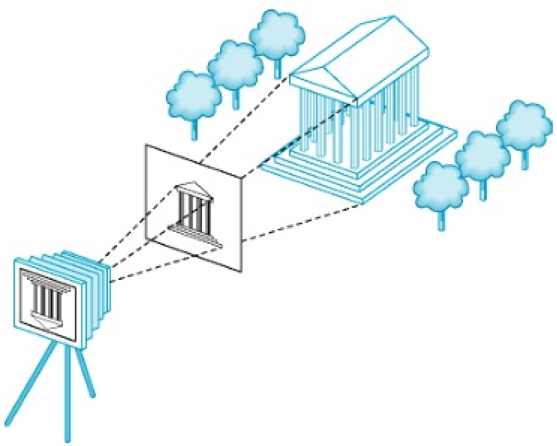
- 1. Position camera : set view matrix
  - lookAt(...)
- 2. Position model : set modeling matrix
  - rotate, translate, scale
- 3. Select lens : set **projection matrix**

Projection matrices

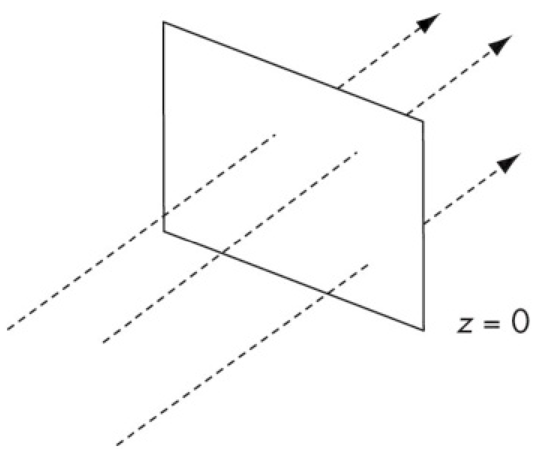


- Viewing and projection are independent
  - View matrix represents camera **position** and **orientation**
  - Projection matrix represents the **shape of the viewing volume**.
    - Field-of-view
    - Aspect ratio
    - Near/far clip planes

2 types of projections



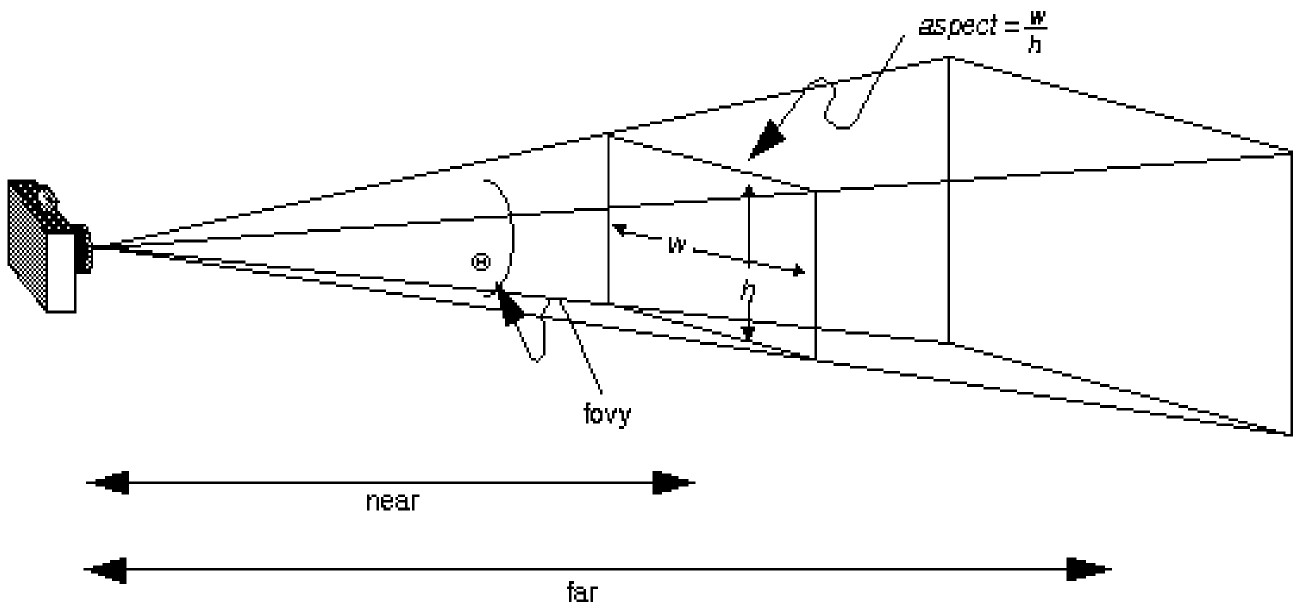
Perspective projection



Parallel projection

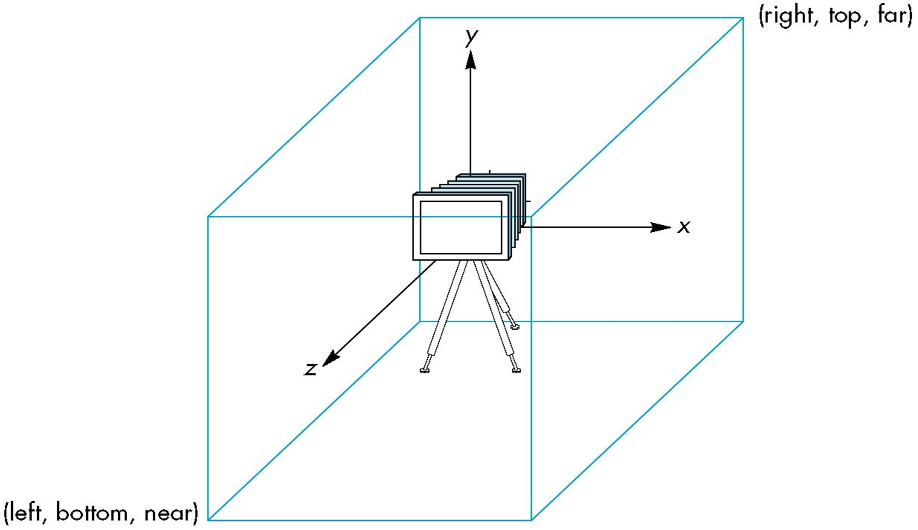
1. Perspective projection
   - Similar to the pinhole camera model.
   - Make the image of far-away objects smaller.
   - Viewing volume : truncated pyramid (frustum).
   - Points are projected along rays through the center-of-projection.
2. Orthographic (or parallel)
   - Viewing volume : ”right parallelepiped”.
   - Bounded by 6 clipping planes : left, right, top, bottom, near, far.
   - Points are projected along rays parallel to the view direction.
   - Most common in 2D applications.
   - The default OpenGL projection.

Perspective projection



- mat4 perspective(float fovy, float aspect, float nearz, float farz)
  - fovy: vertical field-of-view angle
  - aspect: viewport width / height
  - nearz: distance to near clip plane
  - farz: distance to far clip plane
  - Warning: 'near' and 'far' are reserved keywords in C++, don't use them as variable names

Orthographic projection



- mat4 ortho(left, right, bottom, top, near, far)
- If you don't use any projection matrix it is equivalent to using P = I (the identity matrix)
  - This corresponds to left = bottom = near = -1
  - and right = top = far = +1

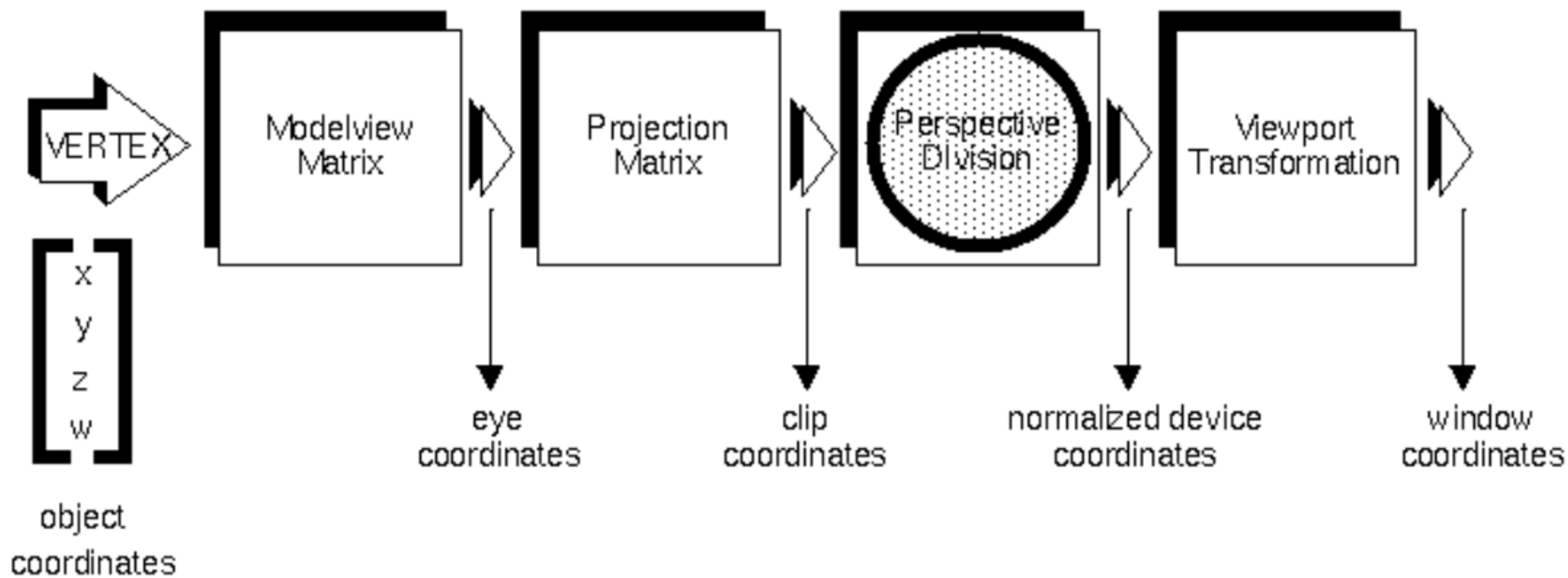
The projection pipeline

- The projection transformation matrix is not an affine matrix: the w component of transformed points may be changed ($w \neq 1$).
- Perspective division restores the w component of homogeneous points:

$$\frac{1}{w}\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

- This is just one part of the full vertex transformation pipeline...

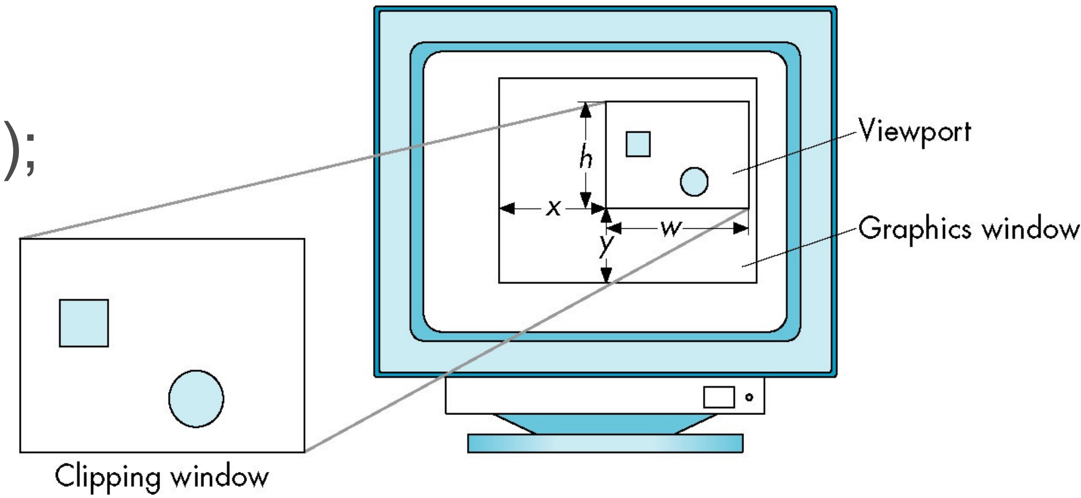
Vertex transformation pipeline



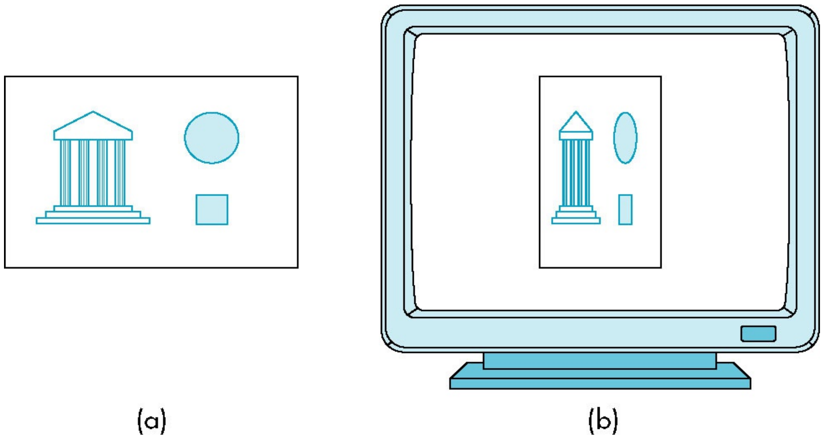
- Clipping is done in clip coordinates.
- Normalized device coordinates range :$(-1 \leq x, y, z \leq 1)$
- Viewport transformation is specified by `glViewport()`.
- Window coordinates (in pixel units) are sent to the rasterizer.

Specifying the viewport transformation

- This is an OpenGL function:
  - glViewport(x, y, width, height);
    - x,y is lower-left corner
    - w,h are width and height



  - You usually want the **aspect ratio** of the viewing volume to match the viewport
  - Otherwise you will get distortion



What if the user resizes the window?

- There is a **glut** callback for that.
- void glutReshapeFunc(void(*func)(int w, int h));
  - Called when the user resizes the window
- The default reshape callback simply calls glViewport(0,0,w,h)
  - If you write your own callback you should call glViewport also;
- If you want to update your projection matrix too this is a good place to do it.
  - aspect ratio of projection matrix should be (float)w/h