Computergrafik

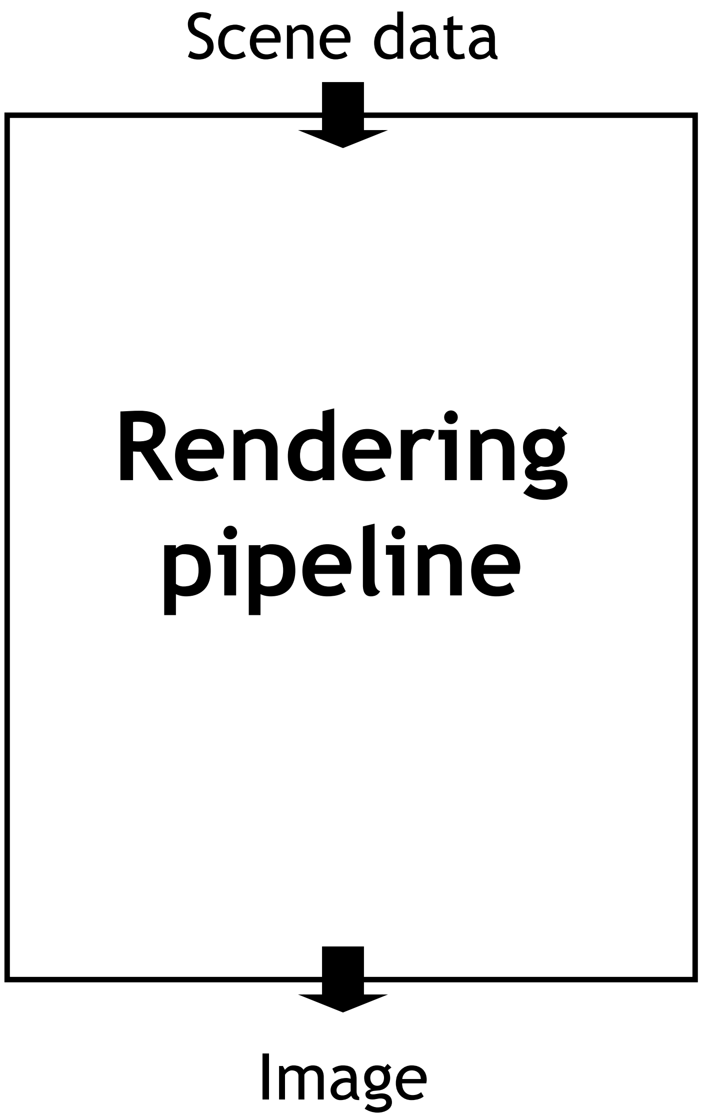
Matthias Zwicker
Universität Bern
Herbst 2016

Today

- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation

Rendering pipeline

http://en.wikipedia.org/wiki/Graphics_pipeline



- Hardware & software that draws 3D scenes on the screen
- Most operations performed by specialized hardware (graphics processing unit, GPU, http://en.wikipedia.org/wiki/Graphics_processing_unit)
- Access to hardware through low-level 3D API (DirectX, OpenGL)
  - jogl is a Java binding to OpenGL, used in our projects http://jogamp.org/jogl/www/
- All scene data flows through the pipeline at least once for each frame (i.e., image)

Rendering pipeline

- Rendering pipeline implements object order algorithm
  - Loop over all objects
  - Draw triangles one by one (rasterization)
- Alternatives?
- Advantages, disadvantages?

Object vs. image order

Object order: loop over all triangles

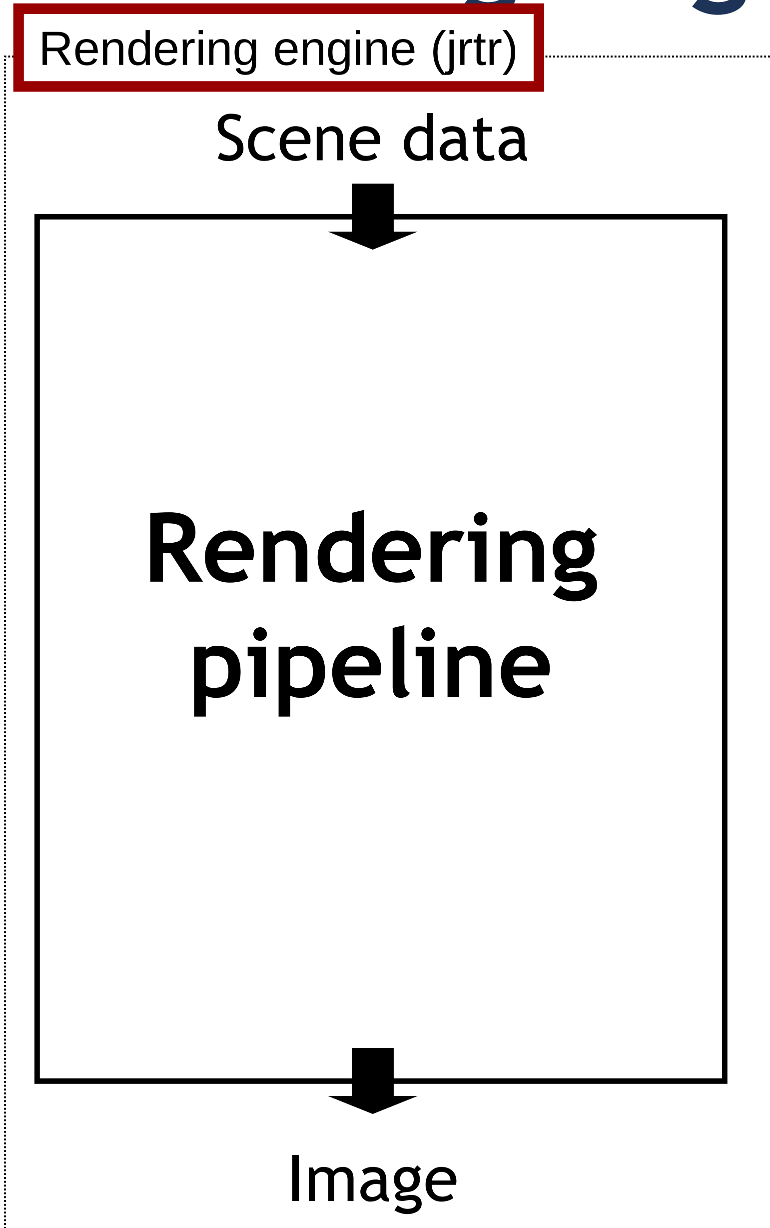
- Rasterization type algorithms
- Desirable memory access pattern („streaming“ scene data one-by-one, data locality, avoid random scene access)
  http://en.wikipedia.org/wiki/Locality_of_reference
- Suitable for real time rendering (OpenGL, DirectX)
- Popular for production rendering (Pixar RenderMan), where scenes often do not fit in RAM
- No global illumination (light transport simulation) with purely object order algorithm

Object vs. image order

Image order: loop over all pixels

- Ray tracing type algorithms
- Undesirable memory access pattern (random scene access)
- Requires sophisticated data structures for fast scene access
- Full global illumination possible
- Most popular for photo-realistic image synthesis

Rendering engine



- Additional software layer ("middle-ware") encapsulating low-level API (OpenGL, DirectX, …)
- Additional functionality (file I/O, scene management, …)
- Layered software architecture common in industry
  - Game engines http://en.wikipedia.org/wiki/Game_engine

Rendering pipeline stages (simplified)

Scene data

↓

Vertex processing, modeling and viewing transformation

↓

Projection

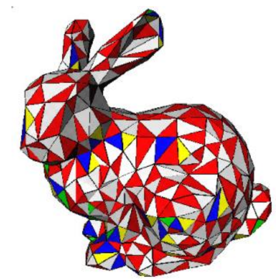
↓

Rasterization, fragment processing, visibility

↓

Image

- Geometry
  - Vertices and how they are connected
  - Triangles, lines, point sprites, triangle strips
  - Attributes such as color



- Specified in object coordinates
- Processed by the rendering pipeline one-by-one

Rendering pipeline stages (simplified)

Scene data

Vertex processing, modeling and viewing transformation

Projection

Rasterization, fragment processing, visibility

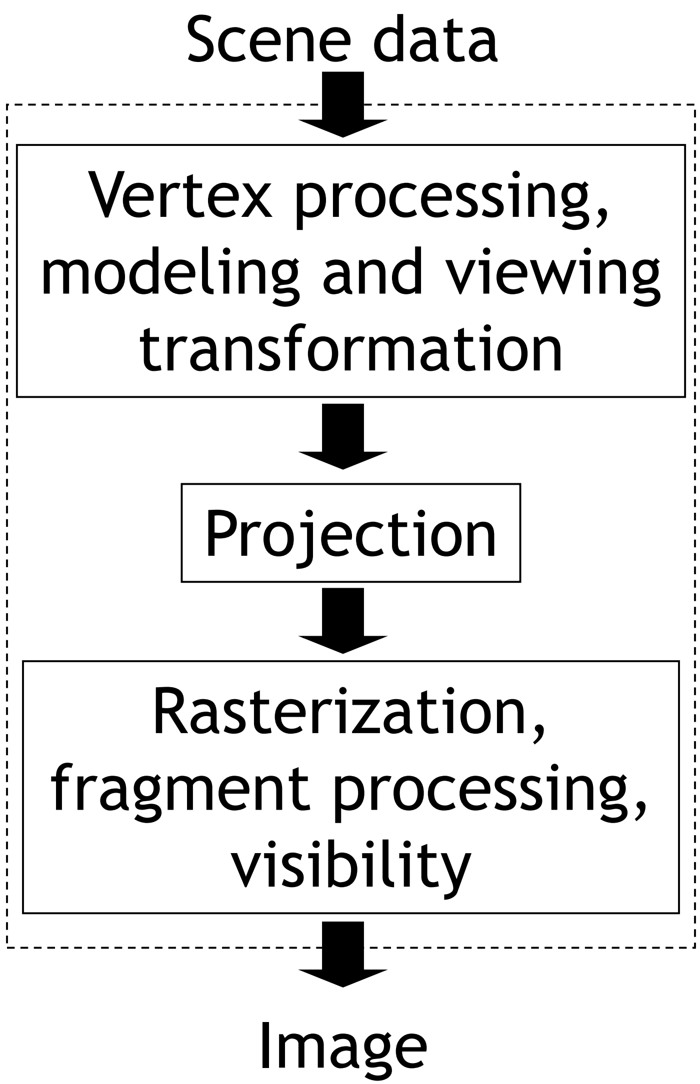
Image

- Transform object to camera coordinates

$$\mathbf{p}_{camera} = \underbrace{\boxed{\mathbf{C}^{-1}\mathbf{M}}}_{\text{MODELVIEW matrix}}\mathbf{p}_{object}$$

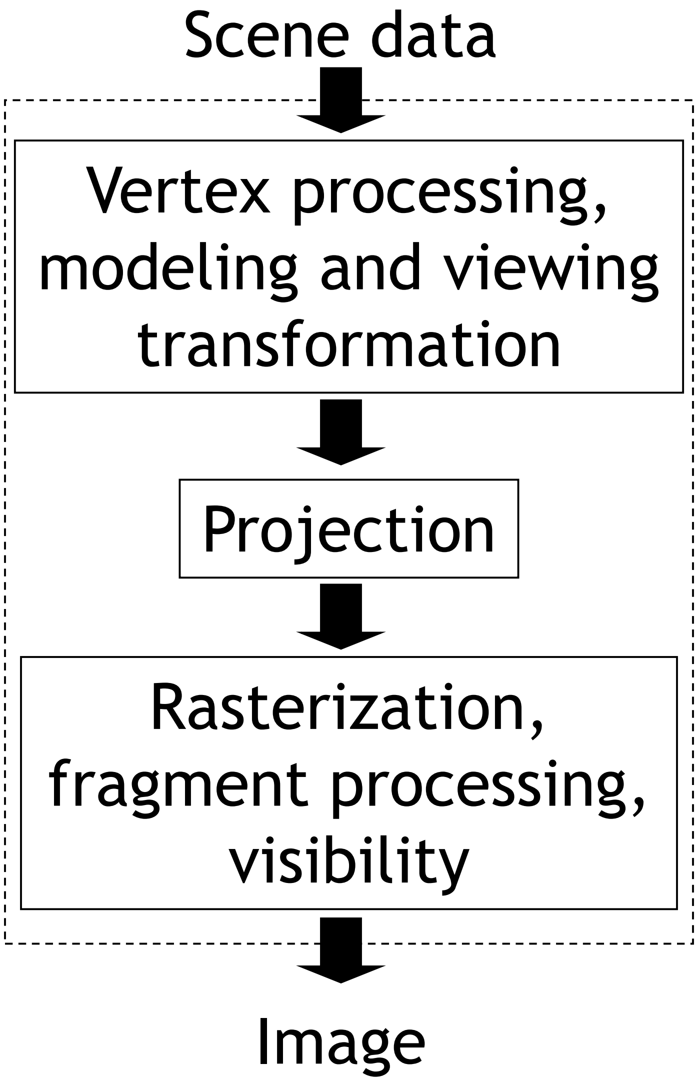
- Additional processing on per-vertex basis
  - Shading, i.e., computing per-vertex colors
  - Deformation, animation
  - Etc.

Rendering pipeline stages (simplified)

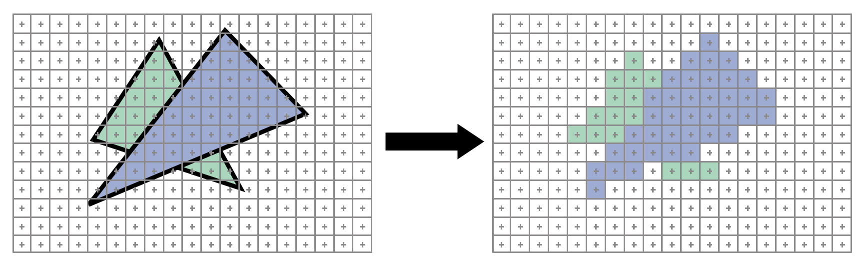


- Project 3D vertices to 2D image positions
- This lecture

Rendering pipeline stages (simplified)

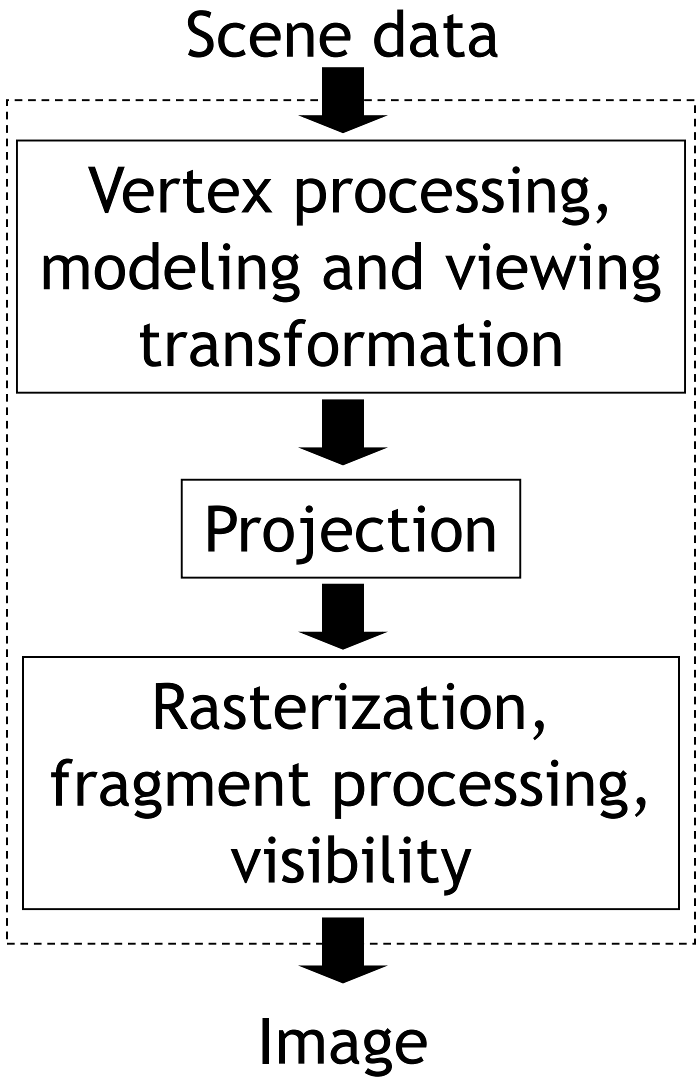


- Draw primitives pixel by pixel on 2D image (triangles, lines, point sprites, etc.)
- Compute per fragment (i.e., pixel) color
- Determine what is visible
- Next lecture



Rasterization

Rendering pipeline stages (simplified)

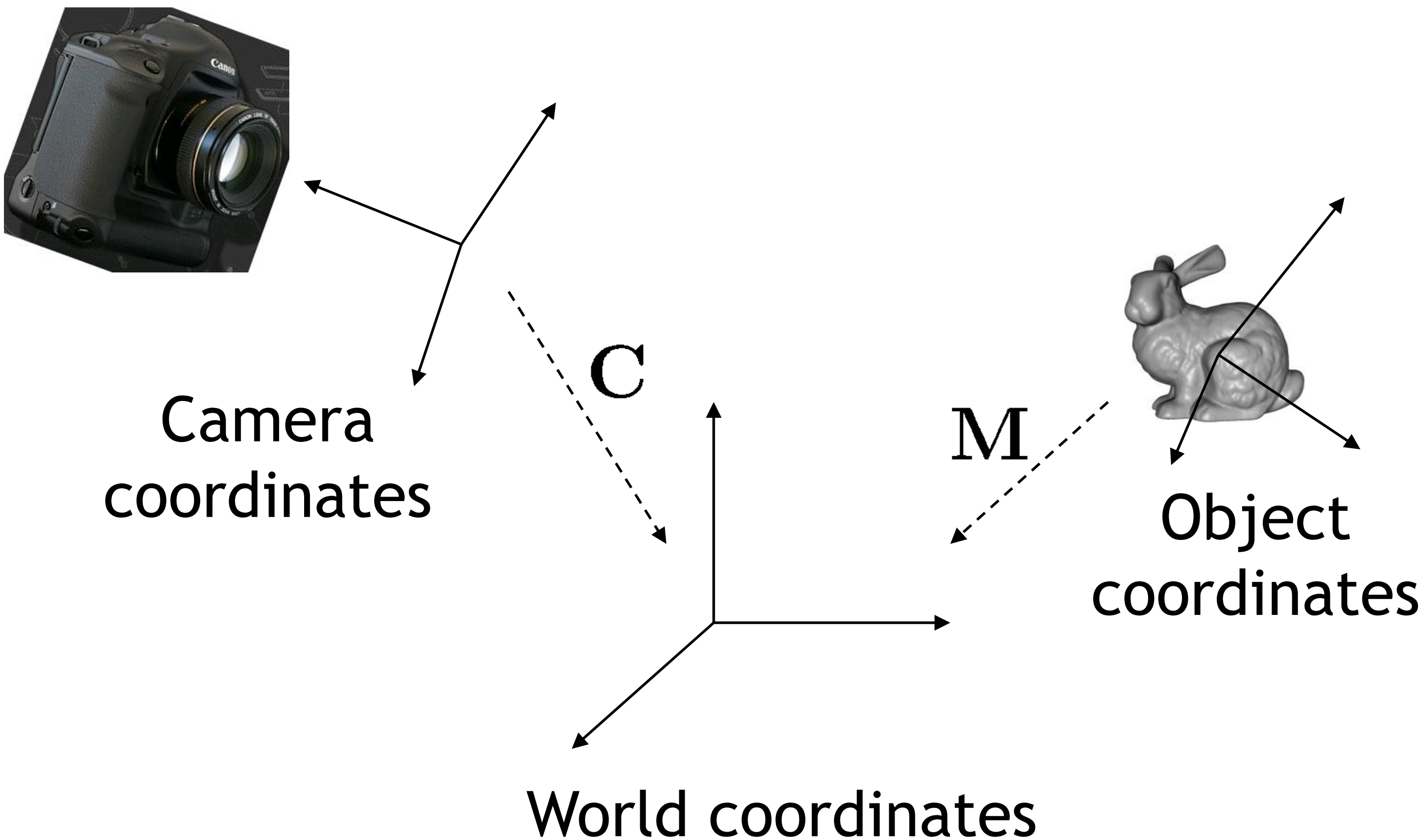


- Grid (2D array) of RGB pixel colors

Today

- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation

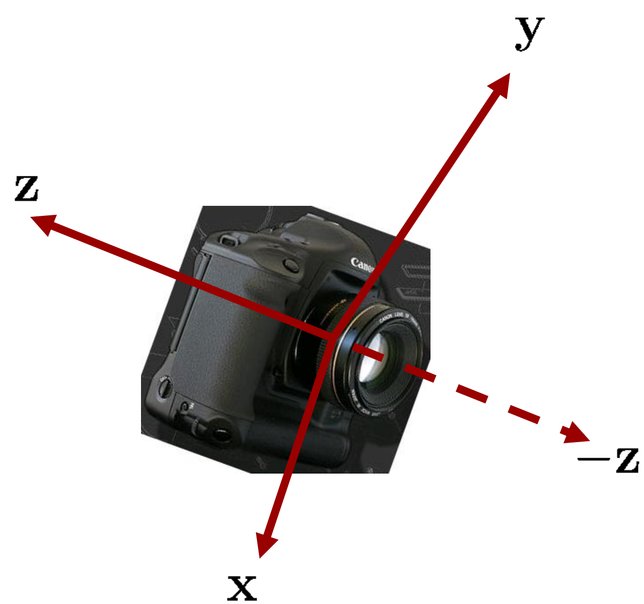
Object, world, camera coords.



$$\mathrm{p}' = \mathbf{C}^{-1}\mathbf{M}\mathrm{p}$$

Objects in camera coordinates

- We have things lined up the way we like them on screen
  - x to the right
  - y up
  - -z going into the screen
  - Objects to look at are in front of us, i.e. have negative z values
- But objects are still in 3D
- Today: how to project them into 2D



Camera coordinates

Projections

- Given 3D points (vertices) in camera coordinates, determine corresponding 2D image coordinates

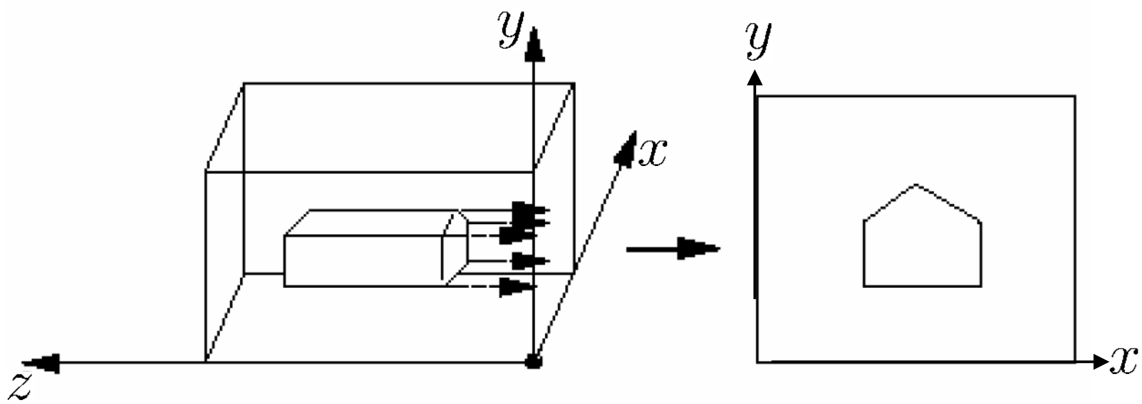
Orthographic projection

- Simply ignore z-coordinate
- Use camera space xy coordinates as image coordinates

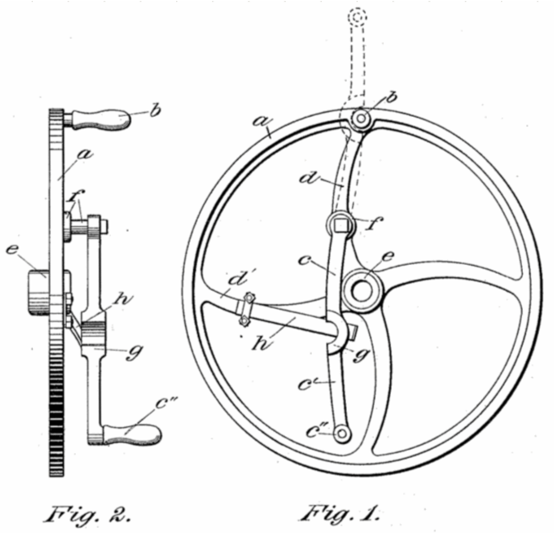
- What we want, or not?

Orthographic projection

- Project points to x-y plane along parallel lines

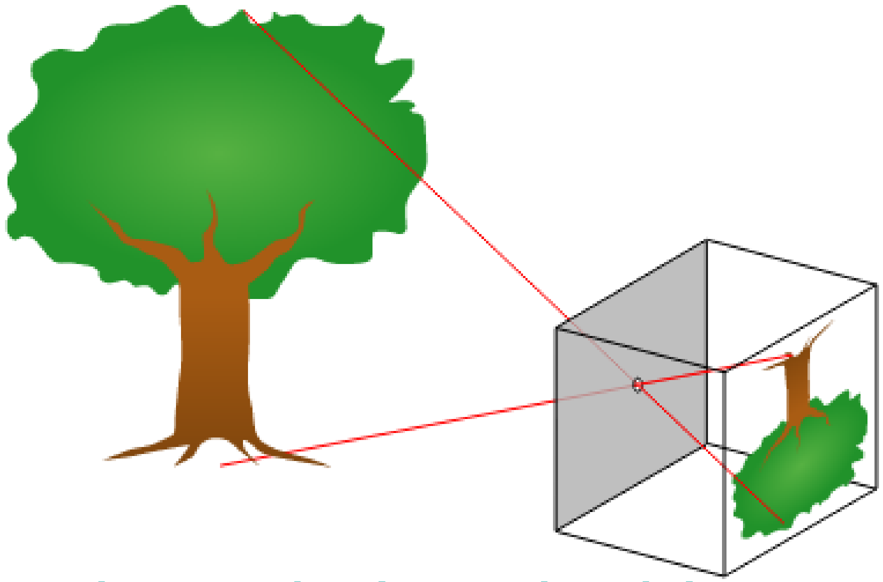


- Graphical illustrations, architecture



Perspective projection

- Most common for computer graphics
- Simplified model of human eye, or camera lens (pinhole camera)
- Things farther away seem smaller

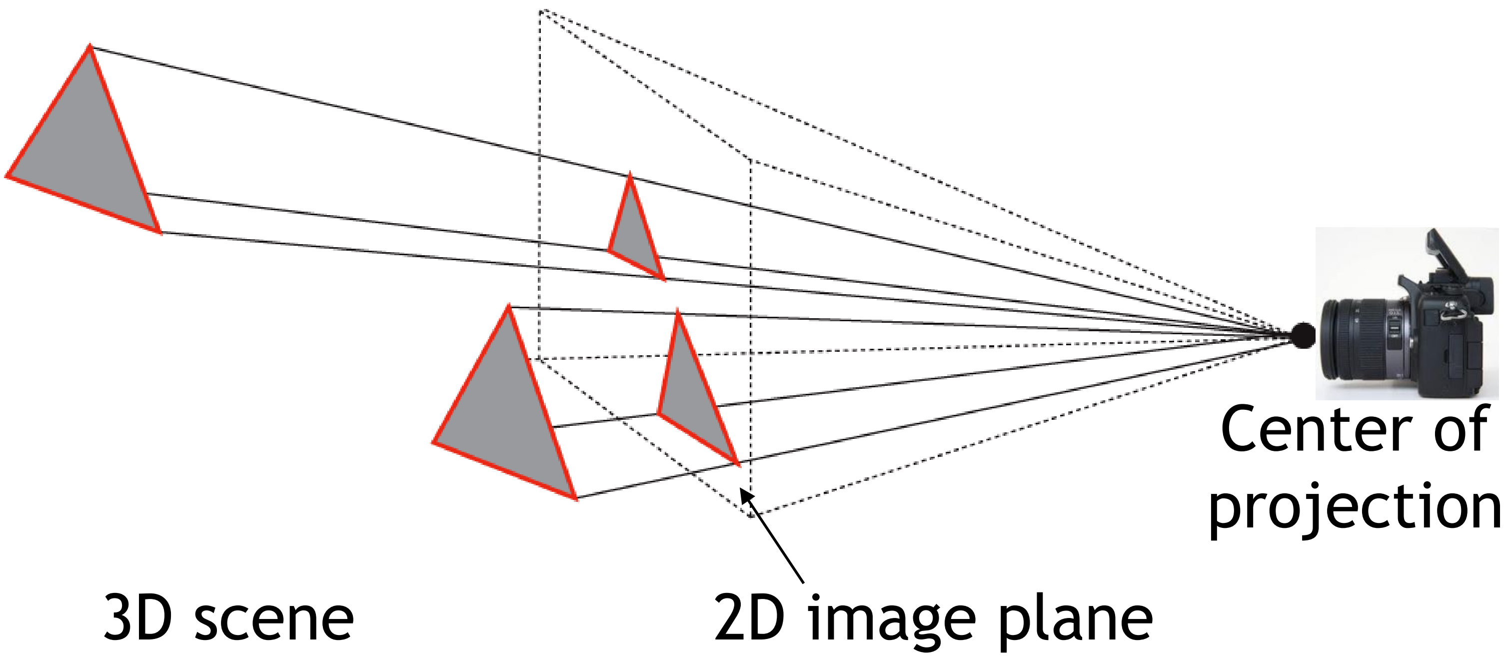


http://en.wikipedia.org/wiki/Pinhole_camera

- Discovery/formalization attributed to Filippo Brunelleschi in the early 1400's

Perspective projection

- Project along rays that converge in center of projection



Perspective projection

Parallel lines
no longer parallel,
converge at one point

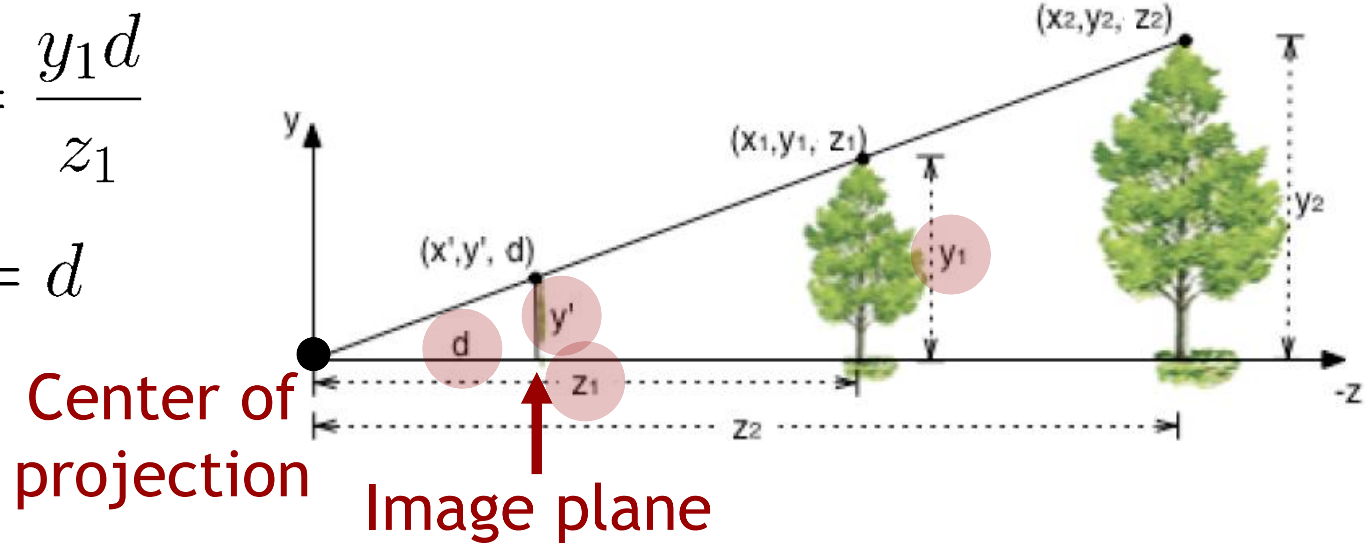
Earliest example
La Trinitá (1427) by Masaccio
http://en.wikipedia.org/wiki/Holy_Trinity_(Masaccio)

Perspective projection

The math: simplified case

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$

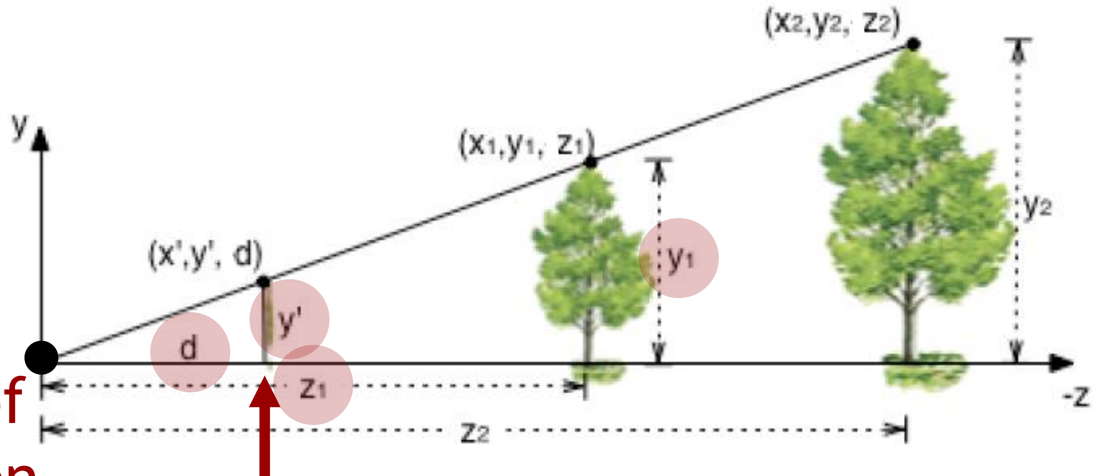


Perspective projection

The math: simplified case

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



Center of projection

Image plane

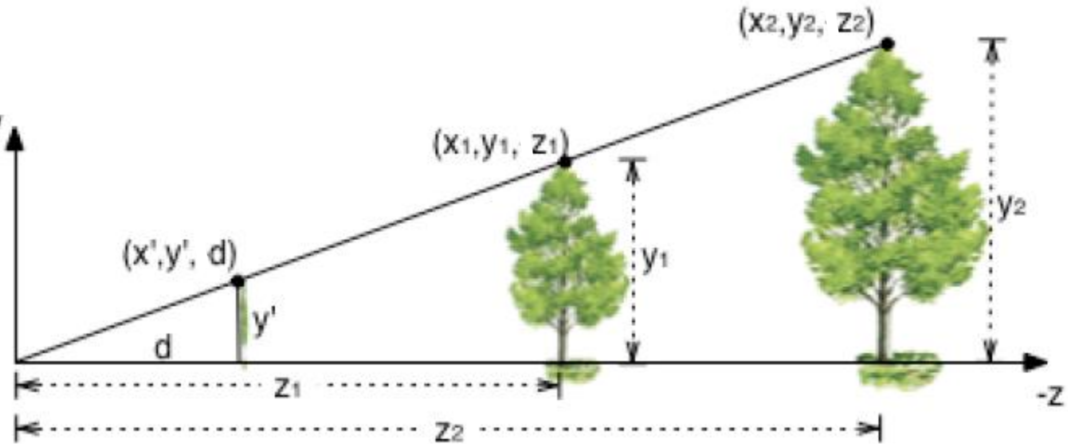
- Can express this using homogeneous coordinates, 4x4 matrices

Perspective projection

The math: simplified case

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \Rightarrow = \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix

Homogeneous coord. != 1!
Homogeneous division

Perspective projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \Rightarrow \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

Projection matrix **Homogeneous division**

- Using projection matrix and homogeneous division seems more complicated than just multiplying all coordinates by d/z, so why do it?
- Will allow us to
  - handle different types of projections in a unified way
  - define arbitrary view volumes

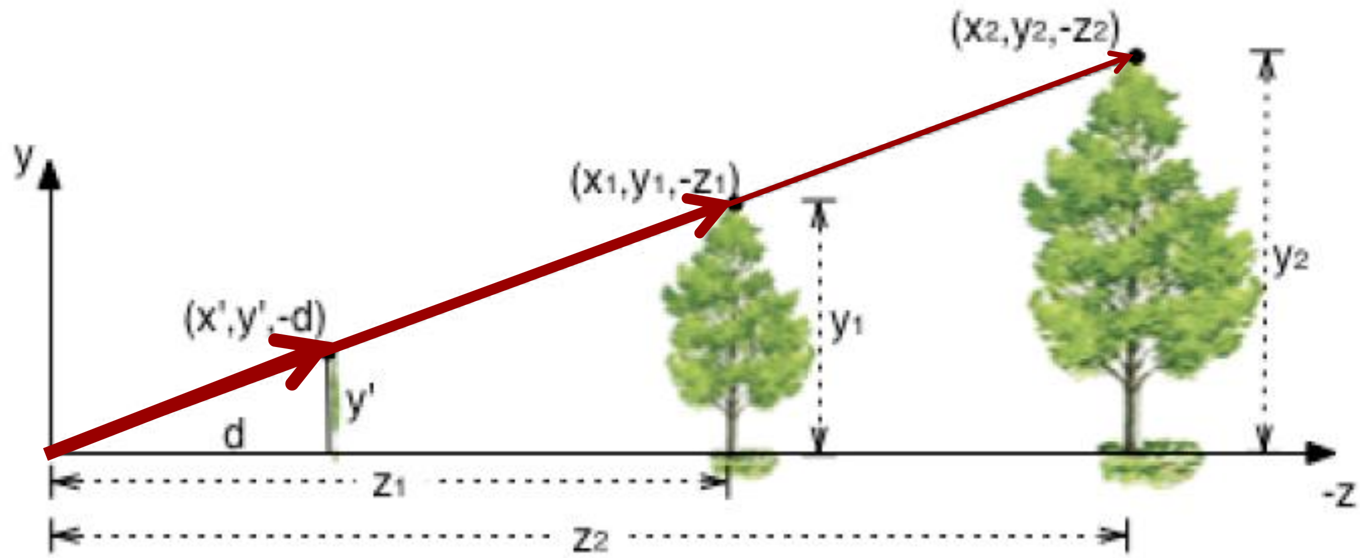
Detour: projective space

http://en.wikipedia.org/wiki/Projective_space

- Projective space: the space of one-dimensional vector subspaces of a given vector space
  - Elements of projective spaces are 1D vector subspaces
  - Each element of 1D subspace is equivalent (represents same element of projective space)

Intuitive example

- All points that lie on one projection line (i.e., a "line-of-sight", intersecting with center of projection of camera) are projected onto same image point
- All 3D points on one projection line are equivalent
- Projection lines form 2D projective space, or 2D projective plane



3D Projective space

- Projective space $\mathbf{P}^3$ represented using $\mathbf{R}^4$ and homogeneous coordinates
  - Each point along 4D ray is equivalent to same 3D point at $w=1$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \sim \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda w \end{bmatrix} \sim \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

„equivalent“

1D vector subspace, arbitrary scalar value λ

Equivalent element, for any λ

3D Projective space

- Projective mapping (transformation):
  any non-singular linear mapping on homogeneous coordinates, for example,

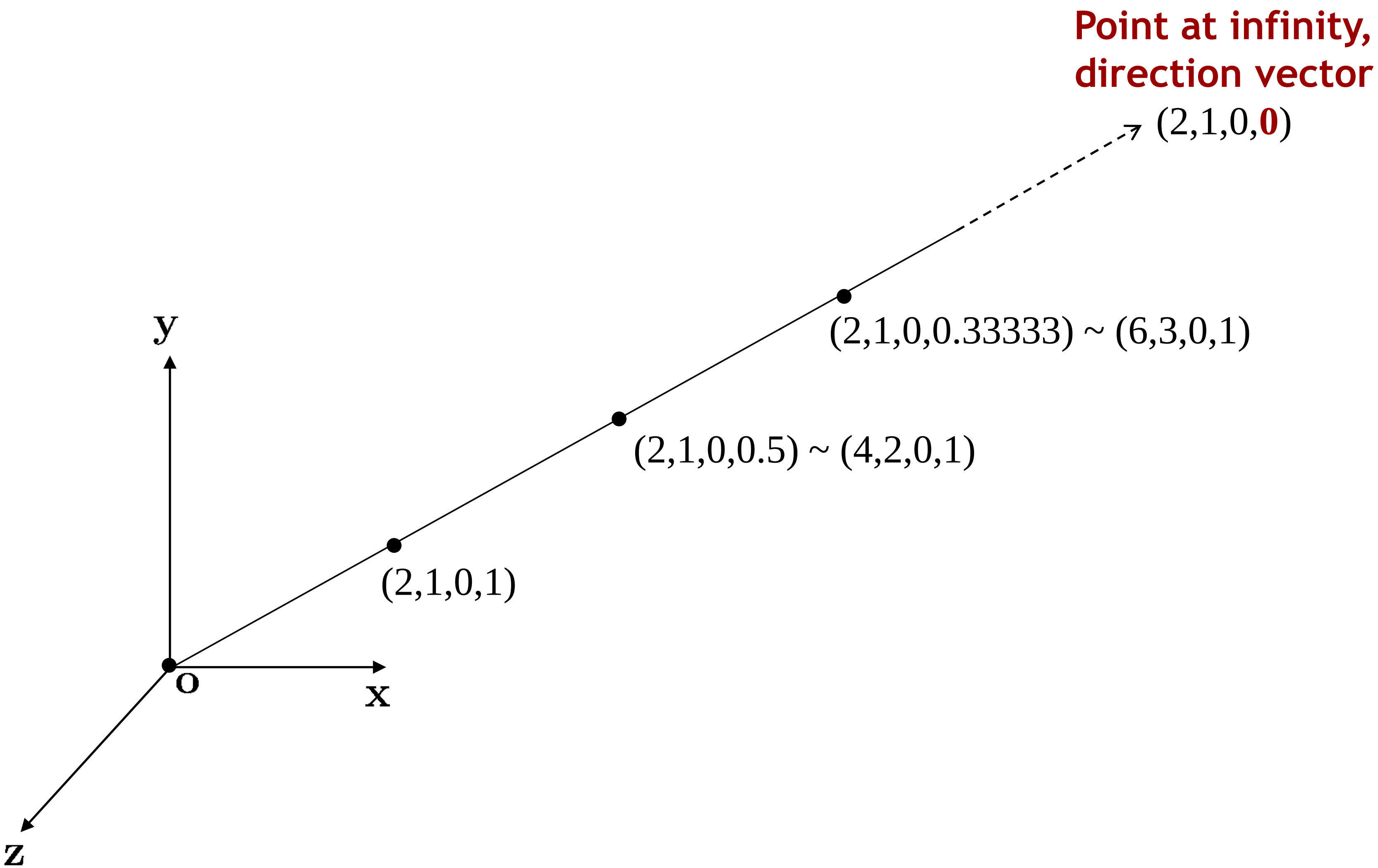
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \sim \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

- Generalization of affine mappings
  - 4th row of matrix is arbitrary (not restricted to [0 0 0 1])
- Projective mappings are collineations
  http://en.wikipedia.org/wiki/Projective_linear_transformation
  http://en.wikipedia.org/wiki/Collineation
  - Preserve straight lines, but not parallel lines
- Much more theory
  http://www.math.toronto.edu/mathnet/questionCorner/projective.html
  http://en.wikipedia.org/wiki/Projective_space

3D Projective space

- $\mathbf{P}^3$ can be interpreted as consisting of $\mathbf{R}^3$ and its „points at infinity“
- Points are said to be at infinity if homogeneous coordinate $w = 0$
  - Represented by direction vector
  - Can actually perform computations with points at infinity (not possible with ∞ sign!)

Points at infinity



2D line intersection

- Do parallel lines intersect at infinity?
  In projective geometry, yes.
  http://www.math.toronto.edu/mathnet/questionCorner/infinity.html

2D line intersection

- Two line equations

$$a_0 x' + b_0 y' + c_0 = 0$$

$$a_1 x' + b_1 y' + c_1 = 0$$

- Intersection: solve two equations in two unknowns

Determinant

$$x_i' = \begin{vmatrix} -c_0 & b_0 \\ -c_1 & b_1 \end{vmatrix} / \begin{vmatrix} a_0 & b_0 \\ a_1 & b_1 \end{vmatrix}$$

$$y_i' = \begin{vmatrix} a_0 & -c_0 \\ a_1 & -c_1 \end{vmatrix} / \begin{vmatrix} a_0 & b_0 \\ a_1 & b_1 \end{vmatrix}$$

- If lines are parallel: division by zero

2D line intersection

- Note: can multiply each of the equations by arbitrary scalar number w, still describes the same line!

$$a_0 x' + b_0 y' + c_0 = 0$$

$$a_0 w x' + b_0 w y' + c_0 w = 0$$

Same line

- Using homogeneous coordinates

$x = wx', y = wy', w$

$$a_0 x + b_0 y + c_0 w = 0$$

Using homogeneous coordinates

- Line equations

$$a_0x + b_0y + wc_0 = 0$$

$$a_1x + b_1y + wc_1 = 0$$

Or equivalent:

$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

$$\begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

Using homogeneous coordinates

- Line equations

$$a_0 x + b_0 y + w c_0 = 0$$

$$a_1 x + b_1 y + w c_1 = 0$$

Or equivalent:

$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

$$\begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

- Intersection: any scalar multiple of

$$\begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} a_0 \\ b_0 \\ c_0 \end{bmatrix} \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$$

- Lines not parallel: intersection $\begin{bmatrix} x_i / w_i \\ y_i / w_i \\ 1 \end{bmatrix} = \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix}$
- Lines parallel: w_i=**0**, intersection at infinity!

Projective space

Projective space

http://en.wikipedia.org/wiki/Projective_space

- [xyzw] homogeneous coordinates
- includes points at infinity (w=0)
- projective mappings (perspective projection)

Vector space

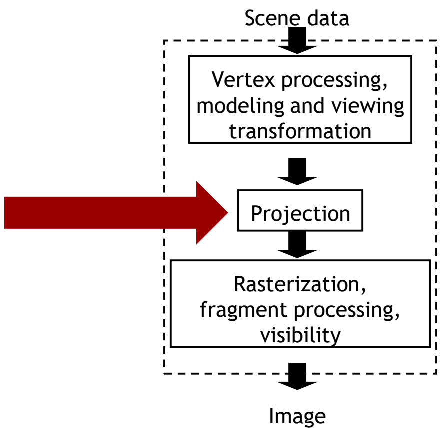
- [xyz] coordinates
- represents vectors
- linear mappings (rotation around origin, scaling, shear)

Affine space

- [xyz1], [xyz0] homogeneous coords.
- distinguishes points and vectors
- affine mappings (translation)

In practice

- Use 4x4 homogeneous matrices like other 4x4 matrices
- Modeling & viewing transformations are affine mappings
  - points keep $w=1$
  - no need to divide by w when doing modeling operations or transforming into camera space
- 3D-to-2D projection is a projective transform
  - Resulting w coordinate not always 1
- Divide by w (perspective division, homogeneous division) after multiplying with projection matrix
  - OpenGL rendering pipeline (graphics hardware) does this automatically



Realistic image formation

- More than perspective projection
- Lens distortions, artifacts

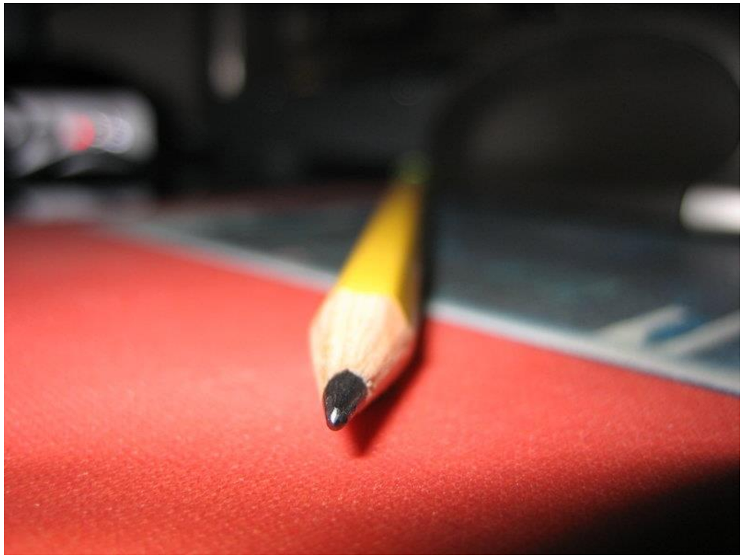
http://en.wikipedia.org/wiki/Distortion_%28optics%29

Barrel distortion

Realistic image formation

- More than perspective projection
- Lens distortions, artifacts
  http://en.wikipedia.org/wiki/Distortion_%28optics%29

Focus, depth of field



Fish-eye lens

http://en.wikipedia.org/wiki/Depth_of_field

Realistic image formation

Chromatic aberration

http://en.wikipedia.org/wiki/Chromatic_aberration

Motion blur

http://en.wikipedia.org/wiki/Motion_blur

- Often too complicated for hardware rendering pipeline/interactive rendering

Today

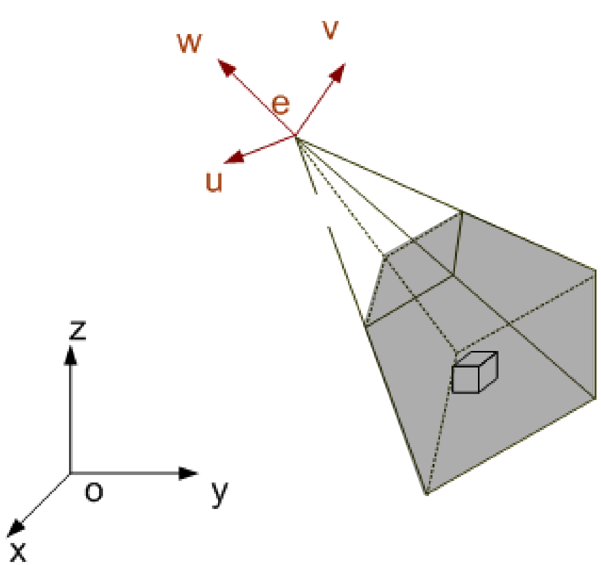
- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation

View volumes

- View volume is 3D volume seen by camera

Perspective view volume

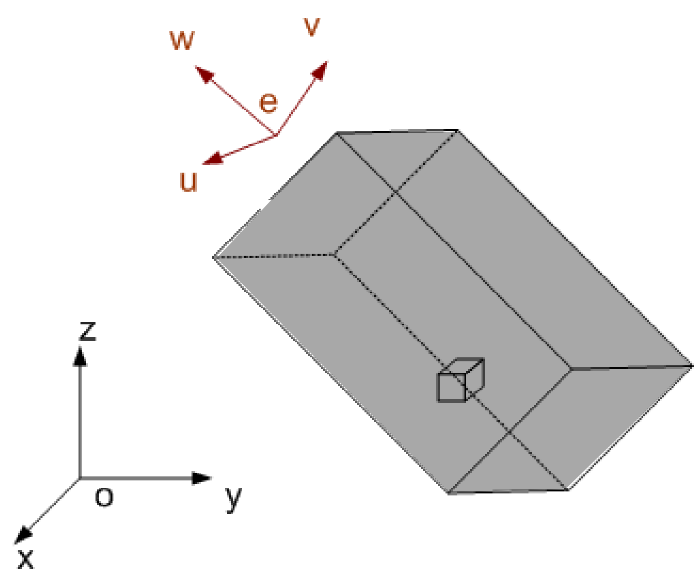
Camera coordinates



World coordinates

Orthographic view volume

Camera coordinates

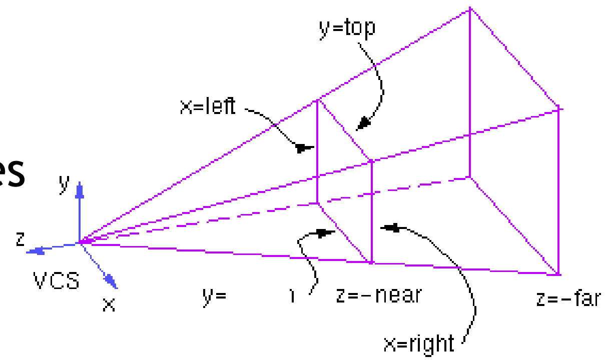


World coordinates

Perspective view volume

General view volume

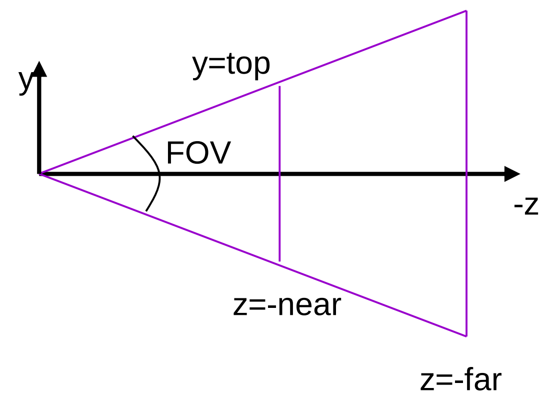
Camera coordinates



- Defined by 6 parameters, in camera coordinates
  - Left, right, top, bottom boundaries
  - Near, far clipping planes
- Clipping planes to avoid numerical problems
  - Divide by zero
  - Low precision for distant objects
- Often symmetric, i.e., left=-right, top=-bottom

Perspective view volume

Symmetric view volume

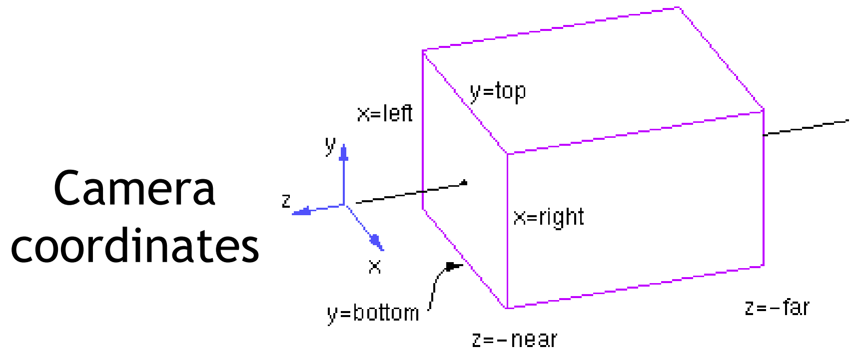


- Only 4 parameters
  - Vertical field of view (FOV)
  - Image aspect ratio (width/height)
  - Near, far clipping planes

$$\text{aspect ratio} = \frac{right - left}{top - bottom} = \frac{right}{top}$$

$$\tan(FOV/2) = \frac{top}{near}$$

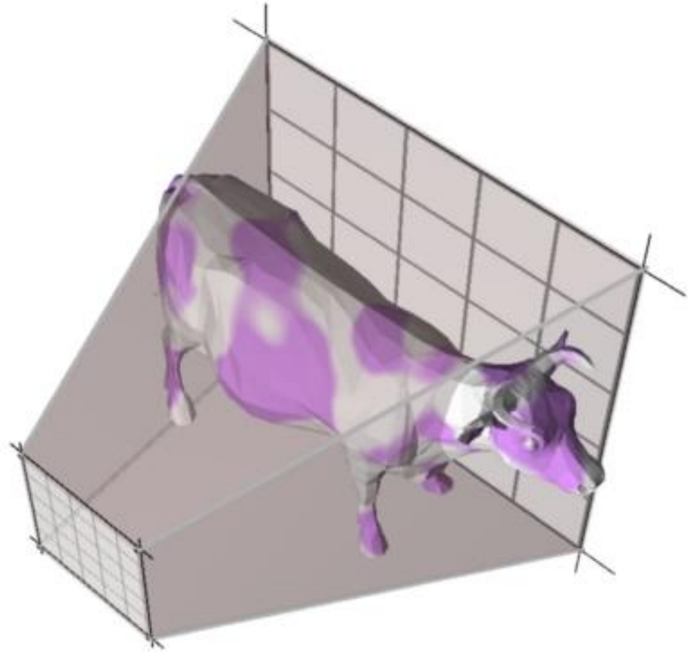
Orthographic view volume



- Parametrized by 6 parameters
  - Right, left, top, bottom, near, far
- If symmetric
  - Width, height, near, far

Clipping

- Need to identify objects outside view volume
  - Avoid division by zero
  - Efficiency, don't draw objects outside view volume



- Performed by OpenGL rendering pipeline
- Clipping always to canonic view volume
  - Cube [-1..1]x[-1..1]x[-1..1] centered at origin
- Need to transform desired view frustum to canonic view frustum

Canonic view volume

- Projection matrix is set such that
  - User defined view volume is transformed into canonic view volume, i.e., unit cube [-1,1]x[-1,1]x[-1,1]

    “Multiplying vertices of view volume by projection matrix and performing homogeneous divide yields canonic view volume, i.e., cube [-1,1]x[-1,1]x[-1,1]“

- Perspective and orthographic projection are treated exactly the same way

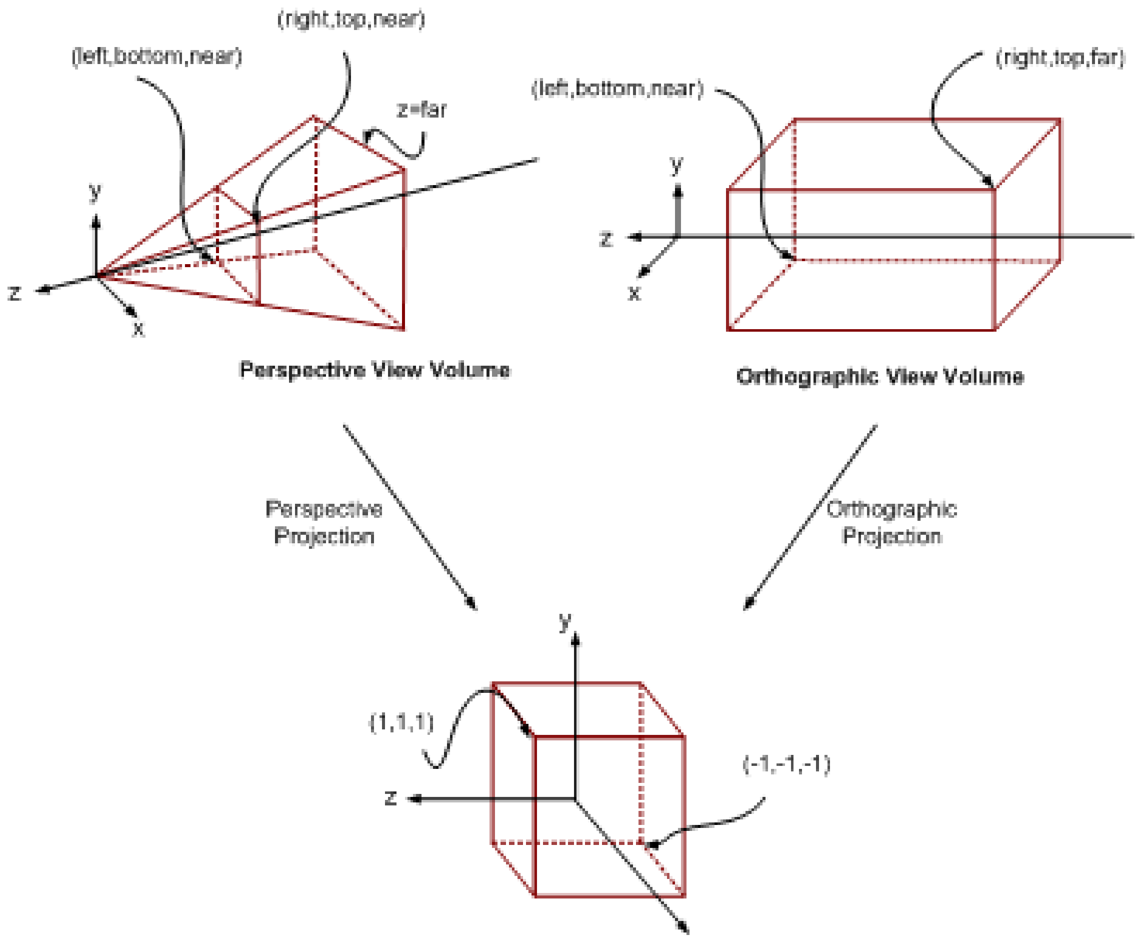
Projection matrix

Camera coordinates

Projection matrix

Canonic view volume

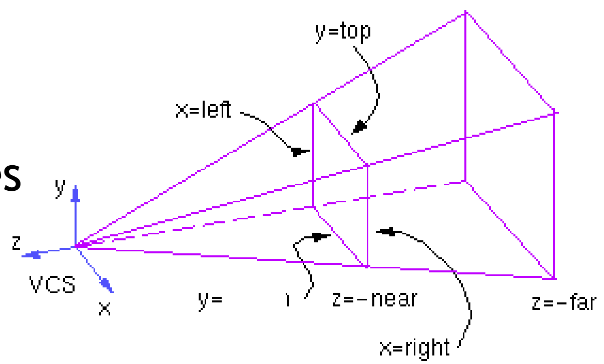
Viewport transformation (later)



Perspective projection matrix

- General view frustum

Camera coordinates



$$\mathbf{P}_{persp}(left, right, top, bottom, near, far) =$$

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

Perspective projection matrix

- Compare to simple projection matrix from before

Simple projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

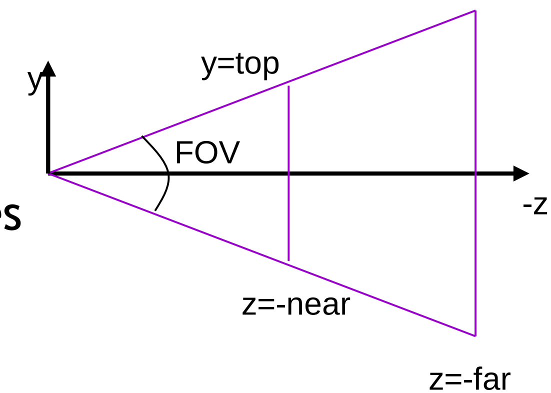
General view frustum

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

Perspective projection matrix

- Symmetric view frustum with field of view, aspect ratio, near and far clip planes

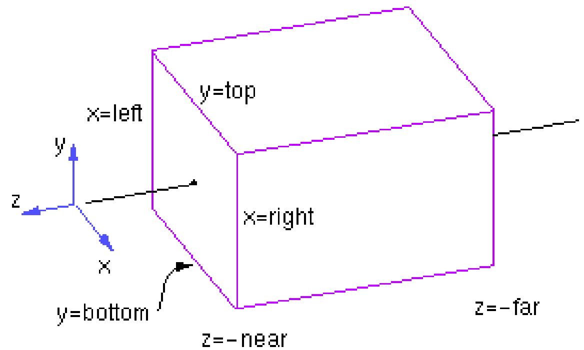
Camera coordinates



$$\mathbf{P}_{persp}(FOV, aspect, near, far) = \begin{bmatrix} \frac{1}{aspect \cdot \tan(FOV/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(FOV/2)} & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2 \cdot near \cdot far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Orthographic projection matrix

Camera coordinates



$$\mathbf{P}_{ortho}(right, left, top, bottom, near, far) = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{far-near} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}_{ortho}(width, height, near, far) = \begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{2}{far-near} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

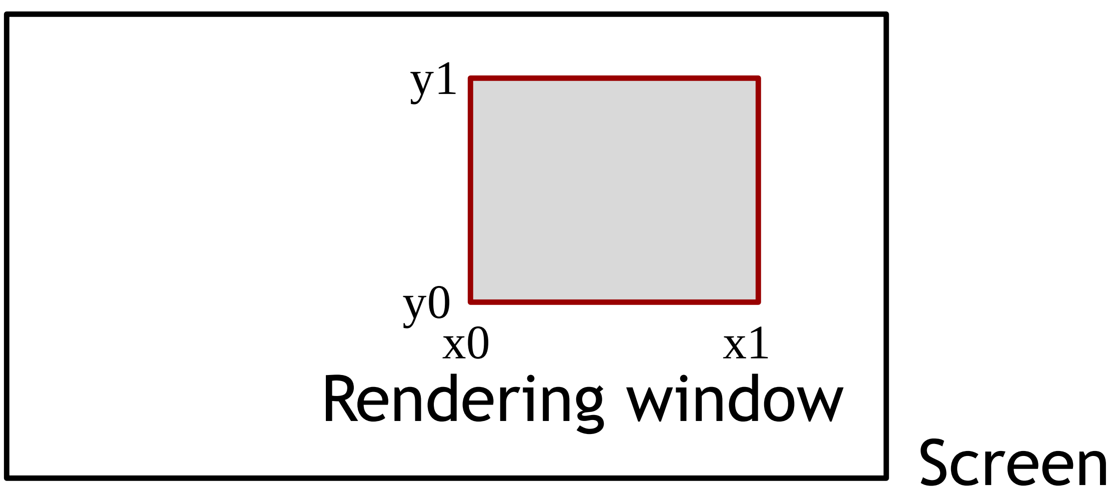
$w = 1$ after mult. with orthographic projection matrix

Today

- Rendering pipeline
- Projections
- View volumes
- Viewport transformation

Viewport transformation

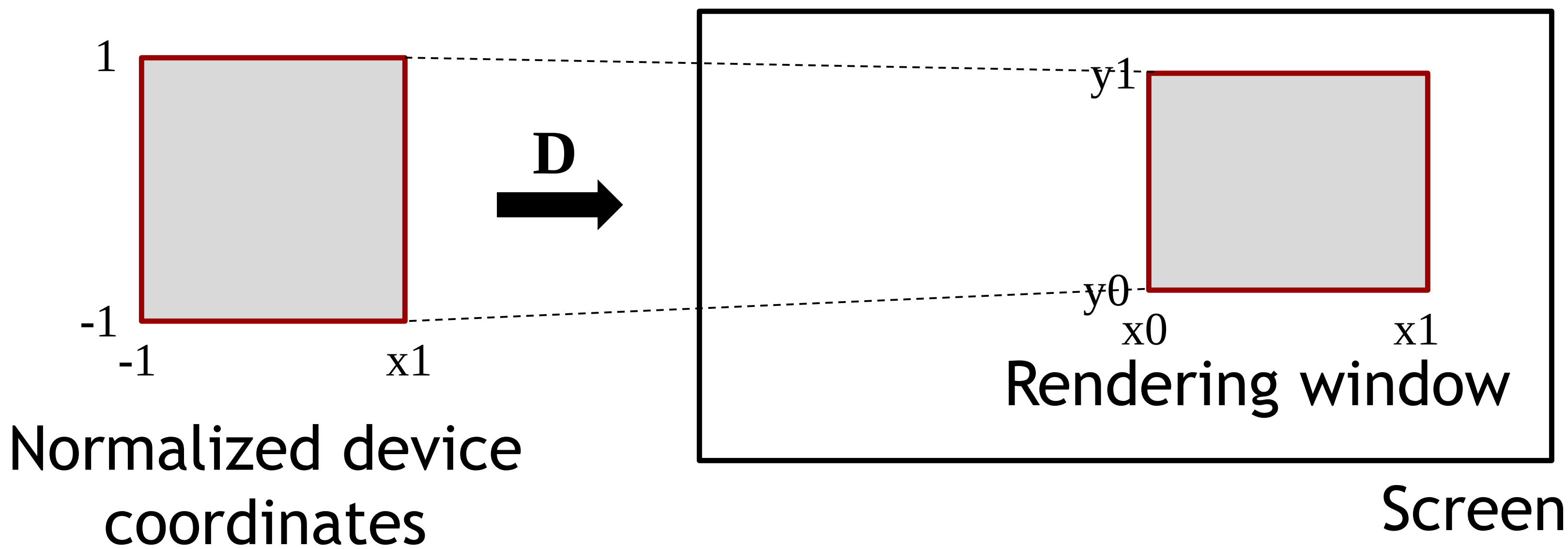
- After applying projection matrix, image points are in normalized view coordinates
  - Per definition range [-1..1] x [-1..1]
- Map points to image (i.e., pixel) coordinates
  - User defined range [x0…x1] x [y0…y1]
  - E.g., position of rendering window on screen



Viewport transformation

- Scale and translation

$$\mathbf{D}(x_0, x_1, y_0, y_1) = \begin{bmatrix} (x_1 - x_0)/2 & 0 & 0 & (x_0 + x_1)/2 \\ 0 & (y_1 - y_0)/2 & 0 & (y_0 + y_1)/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathrm{p}' = \mathbf{DPC}^{-1}\mathbf{M}\mathrm{p}$$

Object space

The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathrm{p}' = \mathrm{DPC}^{-1}\mathrm{M}\mathrm{p}$$

Object space

World space

The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathrm{p}' = \mathrm{DPC}^{-1}\mathrm{Mp}$$

Object space

World space

Camera space

The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathrm{p}' = \mathrm{D}\,\mathrm{P}\,\mathrm{C}^{-1}\,\mathrm{M}\,\mathrm{p}$$

Object space

World space

Camera space

Canonic view volume

The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathrm{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathrm{p}$$

Object space

World space

Camera space

Canonic view volume

Image space

The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{C}$, viewport matrix $\mathbf{D}$

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

Pixel coordinates $\begin{matrix} x'/w' \\ y'/w' \end{matrix}$

OpenGL details

- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{P}$, viewport matrix $\mathbf{D}$

Model-view matrix

$$\mathbf{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathbf{p}$$

Projection matrix

- OpenGL rendering pipeline performs these matrix multiplications in vertex shader program
  - More on shader programs later in class
- User just specifies the model-view and projection matrices
- See Java code jrtr.GLRenderContext.draw and default vertex shader in file default.vert

OpenGL details

- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$, projection matrix $\mathbf{P}$, viewport matrix $\mathbf{D}$

Model-view matrix

$$\mathrm{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathrm{p}$$

Projection matrix

- Exception: viewport matrix, **D**
  - Specified implicitly via glViewport()
  - No direct access, not used in shader program

Coming up

Next lecture

- Drawing (rasterization)
- Visibility (z-buffering)

Exercise session

- Project 2, interactive viewing