## Lecture \#7

# Viewing and Perspective 

Computer Graphics
Winter Term 2020/21

Marc Stamminger / Roberto Grosso

## Remember: Mapping 3D to 2D

- Simple projection: parallel projection onto plane
- Affine $\rightarrow$ parallel lines remain parallel

orthographic projection
- real perspective $\rightarrow$ poirt projection

"real" perspective projection


## Perspective Projection

- Project scene onto image plane using point projection with camera as projection center



## Perspective Projection

- Strategy based on simple mathematical rule
- Project objects directly towards the eye
- Draw object where they meet a view plane in front of the eye



## Perspective Projection



Albrecht Dürer
Der Zeichner der Laute
1512-1525

## Albrecht Dürer

Der Zeichner des liegenden Weibes 1512-1525


## Perspective Projection

- Linear Perspective Projection: The pinhole camera
illum in tabula per radios Solis, quàm in ceelo contingit: hoc eft, fi in ccelo fuperior pars deliquiũ patiatur, in radiis apparebit inferior deficere, vt ratio exigit optica.


Sic nos exactc̀ Anno.1544. Louanii celipfim Solis obferuauimus, inuenimuś́; deficere paulò plus $\not$ đ̈ dex-
"When images of illuminated objects ... penetrate through a small hole into a very dark room ... you will see [on the opposite wall] these objects in their proper form and color, reduced in size ... in a reversed position, owing to the intersection of the rays".
Da Vinci
http://www.acmi.net.au/AIC/CAMERA_OBSCURA.html (Russell Naughton)

## Perspective Projection



Pietro Perugino, fresco at the Sistine Chape (1481-82). Source: http://en.wikipedia.org/wiki/Vanishing_point


Canaletto 1735-45. The Piazza of San Marco, Venice. One point perspective

## Source:

http://www.siggraph.org/education/materials/HyperGraph/viewing/view3d/perspect.htm

## Perspective Projection

- Properties:
- Objects appear smaller as their distance to the observer increases (foreshortening)
- Vanishing points (Fluchtpunkte): Lines parallel in world converge to a single point in image space (rails of a railroad)
- 1, 2 or 3 -point perspective: Lines parallel to 1,2 or 3 of the main axes converge in a vanishing point, the others remain parallel
- One point perspective
- the image plane is orthogonal to one of the coordinate axis and parallel to the other two.
- Two point perspective
- The image plane in parallel to one coordinate axis and intersect the other two.
- Three-point perspective
- The image plane intersects all three coordinate axis.


## Perspective Projection

- One-point perspective - one vanishing point

source: http://stevewebel.com/photographer/wp-content/uploads/2008/04/vanishing-point.jpg

source:http://cavespirit.com/CaveWall/5/vanishin g_point_high_horizon.jpg


## Perspective Projection

- Two-point perspective - two vanishing points

http://www.vintage-views.com/WaresModernPerspective/images/1219k6-Plate1.jpg


## Perspective Projection

- Two-point perspective - two vanishing points


Sanaa-essen-Zollverein-School-of-Management-and-Design-220409-01.jpg de.wikipedia.org

## Perspective Projection

- orthographic projection


Computer Graphics 2020/2 VI

- perspective projection



## Perspective Projection

- How can we describe this projection?
- Look at special case:
- camera in origin
- looks into z-direction
- projection onto $z=1$ plane



## Perspective Projection

- How can we handle this ?
- Remember homogeneous coordinates:

$$
\binom{x}{y} \rightarrow\left(\begin{array}{l}
x \\
y \\
1
\end{array}\right) \stackrel{A}{\rightarrow}\left(\begin{array}{c}
x^{\prime} \\
y^{\prime} \\
w^{\prime}
\end{array}\right) \rightarrow\binom{\frac{x^{\prime}}{w^{\prime}}}{\frac{y^{\prime}}{w^{\prime}}}
$$

- $w$ is common divisor
- if we move $z$ to $w$, the final division will generate perspective:

$$
\left(\begin{array}{l}
x \\
y \\
z
\end{array}\right) \rightarrow\left(\begin{array}{l}
x \\
y \\
z \\
1
\end{array}\right) \xrightarrow{\rightarrow}\left(\begin{array}{l}
x \\
y \\
z \\
z
\end{array}\right) \rightarrow\left(\begin{array}{c}
x / z \\
y / z \\
1
\end{array}\right) \quad \text { with } M=\left(\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0
\end{array}\right)
$$

## Viewing and Projection

- How can we generalize all this?
$\rightarrow$ To describe both orthogonal and perspective projection, we consider two separate steps, both described as matrices:
- First Viewing
- defines camera position and view direction
- rigid transformation
- moves camera position to origin and aligns axes:
$\rightarrow x$-axis points in horizontal image direction
$\rightarrow y$-axis points in vertical image direction
$\rightarrow z$-axis points in view direction
- to get a right-handed coordinate system, often $-z$ is view direction
- Then Projection
- then an orthogonal or perspective projection is performed
- and a rectangular regions from the image plane mapped to the final image


## Viewing Transformation

- Compute axes for viewing transformation $\rightarrow u$-axis points in horizontal image direction
$\rightarrow v$-axis points in vertical image direction
$\rightarrow w$-axis points in view direction

- We define these indirectly using the following three more intuitive vectors:
- Eye position $e$ : location of the eye / center of the lens
- Gaze direction $g$ : direction the viewer is looking
- View-up vector $t$ : points upwards
$\rightarrow$ vertical in image
$\rightarrow$ typically $(0,1,0) \rightarrow$ "y is up" or $(0,0,1) \rightarrow$ " $z$ is up"



## Viewing Transformation

- Usually, we think in right-handed coordinate systems
- but $u, v, w$ on the previous slide are left-handed
- if we want to maintain the meaning of $u$ and $v$ (right and up), we have to flip the $z$-axes and make $-z$ to the view direction.



## Viewing Transformation

- Given: camera position $e$, view direction $g$ and up-vector $t$
- Compute new basis: origin $e$ and basis vectors ( $u, v, w$ )
- $w$
- points opposite to gaze direction ("-z" convention): $w=-g /\|g\|$
- $v$
- almost the same as $t$, but not always
- if gaze direction is not perpendicular to $t$, then we have to rotate $v$ away from $t$
- $v, t$, and $g$ should be in one plane
- simple solution: first compute $u$, then $v$
- $u$
- should be perpendicular to both $g$ and $t$ :

$$
u=\frac{t \times w}{\|t \times w\|}
$$

- then $v$ is perpendicular to both $u$ and $w$ :


$$
\mathrm{v}=\mathrm{w} \times u
$$

## Viewing Transformation

- Given: camera position $e$, look-at point $a$ and up-vector $t$
- Recipe
- $g=a-e$
- $w=-g /\|g\|$
- $u=\frac{t \times w}{\|t \times w\|}$
- $\mathrm{v}=\mathrm{w} \times u$
- The viewing transformation is then (see intro slides; $u, v, w$ are orthonormal):
$\cdot R=\left(\begin{array}{lll}u_{x} & v_{x} & w_{x} \\ u_{y} & v_{y} & w_{y} \\ u_{z} & v_{z} & w_{z}\end{array}\right) \quad \mathrm{e}=\left(\begin{array}{l}e_{x} \\ e_{y} \\ e_{z}\end{array}\right)$
- $M_{v}=\left(\begin{array}{cccc} & & & \vdots \\ & R^{T} & & -R^{T} e \\ & & & \vdots \\ 0 & 0 & 0 & 1\end{array}\right)=\left(\begin{array}{cccc}u_{x} & u_{y} & u_{z} & -u^{T} e \\ v_{x} & v_{y} & v_{z} & -v^{T} e \\ w_{x} & w_{y} & w_{z} & -w^{T} e \\ 0 & 0 & 0 & 1\end{array}\right)$


## Viewing $\rightarrow$ Projection

- When the coordinates are aligned with the camera, we have a much simpler situation:



## Orthogonal Projection

- Projection onto image plane $z=0$ :

$$
M=\left(\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{array}\right)
$$

- This way, z (=depth) gets lost...
- so we keep $z$ :

$$
M_{\text {ortho }}=\left(\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right)=I
$$

## Perspective Projection

- For a perspective projection, we use the $z=1$ image plane

$$
M=\left(\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0
\end{array}\right)
$$

- Again, $z$ gets lost $\left(z^{\prime}=z / z=1\right)$
- We thus use:

$$
M_{\text {perspective }}=\left(\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -1 \\
0 & 0 & 1 & 0
\end{array}\right)
$$

- now $z \rightarrow \frac{z-1}{z}=1-\frac{1}{z}$
- new depth not linear in $z$, but order is maintained


## Perspective Projection

- Perspective matrix maps infinite view frustum to a box !
- after this mapping, $(x, y)$ are image coordinates and $z$ is depth
- $z$ has non-linear to depth



## Cropping

- After projection (both orthogonal and perspective)
- $x$ and $y$ are image coordinates, $z$ is depth
- Finally, we have to define
- which window of this image plane becomes our final image
- this image is a rectangular interval $\left[x_{\min }, x_{\max }\right] \times\left[y_{\min }, y_{\max }\right]$
- usually: $x_{\min }=-x_{\max }, y_{\min }=-y_{\max }$




## Cropping

- to this end, we map $\left[x_{\min }, x_{\max }\right] \times\left[y_{\min }, y_{\max }\right]$ to $[-1,1]^{2}$ :

$$
M_{c}=\left(\begin{array}{cccc}
\frac{2}{x_{\max }-x_{\min }} & 0 & 0 & -\frac{x_{\max }+x_{\min }}{x_{\max }-x_{\min }} \\
0 & \frac{2}{x_{\max }-x_{\min }} & 0 & -\frac{y_{\max }+y_{\min }}{x_{\max }-x_{\min }} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{array}\right)
$$

## Depth Normalization

- finally, we also want to normalize depth to be in $[-1,1]$
- we define a near plane and a far plane: $z=-z_{\text {near }}$ and $z=-z_{f a r}$ (remember: „-z"-convention)
- linear mapping on $z$, such that $z_{\text {near }} \rightarrow-1$ and $z_{\text {far }} \rightarrow 1$ :

$$
M_{c}=\left(\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & A & B \\
0 & 0 & 0 & 1
\end{array}\right)
$$

- choose $A$ and $B$, such that
- $z=-n$ gets mapped to $z=-1$ and
- $z=-f$ gets mapped to $z=1$
$\rightarrow A=-\frac{f+n}{f-n}, B=-\frac{2 f n}{f-n}$


## 

- Standard projection, cropping, and depth normalizationare merged to a single matrix, called the Projection Matrix
- Orthogonal Projection:

The image window is defined by ( $l, r, b, t$ ) (left,right,bottom,top) and the depth range by $n$ and $f$


- $M_{\text {ortho }}(l, r, b, t, n, f)=\left(\begin{array}{cccc}\frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1\end{array}\right)$


##  ERLANGEN-NÜRNBERG

- Perspective Projection:

The depth range goes from $n$ to $f$ (near to far), and the image window ( $l, r, b, t$ ) is defined on the near-plane


- $M_{\text {perspective }}(l, r, b, t, n, f)=\left(\begin{array}{cccc}\frac{2 n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2 n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2 f n}{f-n} \\ 0 & 0 & -1 & 0\end{array}\right)$


## 

- Perspective Matrix usually defined by
- opening angle in y-direction: field of view in $y \rightarrow f o v y$
- aspect ratio: ration of width over height $\rightarrow$ aspect
- near and far plane $\rightarrow n, f$

- $-l=r=\operatorname{aspect} \cdot n \cdot \tan \frac{\text { fovy }}{2}$
- $-b=t=n \cdot \tan \frac{f o v y}{2}$
- Large field of view corresponds to a wide angle lens, small field of view to a tele lens


## Projection of View Frusta

- The projection matrices transform the orthogonal and perspective view frustum into the canonical view frustum $[-1,1]^{3}$

perspective matrix



## Demo Viewing and Perspective


Mouse modifies
Model $\bigcirc$ View Projection

Space

## Normalizing Transformation

- The perspective matrix transforms the view frustum to the unit cube

- We also call this the Normalizing Transformation
- It belongs to the class of Projective Transformations


## Normalizing Transformation

- The perspective matrix transform the view frustum to the unit cube

- Regions close to observer are enlarged, distant regions are shrunk $\Rightarrow$ perspective distortion



## Normalizing Transformation

- homogenous coordinates allows us to represent points at infinity: $\lim (\mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{w})=$ point at infinity in direction $(x, y, z)$

$$
w \rightarrow 0
$$

$\rightarrow$ points at infinity $=$ directions $=(x, y, z, 0)$

- A projective matrix can map such infinity points $(x, y, z, 0)$ to finite points $(x, y, z, w), w \neq 0$ and vice versa !
- Intersection of parallel lines = point at infinity = direction of these lines gets mapped to finite point and vice versa





## Normalizing Transformation

- Properties:
- lines remain lines
- parallel lines don't remain parallel
- ratios are not preserved



## Normalizing Transformation

- How to choose near and far planes: Nonlinear mapping of $z$ :

$$
z \rightarrow \frac{n+f}{f-n}-\frac{2 n f}{(f-n) z}
$$

- $z$-buffer with low resolution
- Objects at far distance collapse
- Drawing will fail sorting far objects due to insufficient resolution, z-fighting.



## Normalizing Transformation

- Objects at far distance collapse



## Normalizing Transformation

- choose reasonable near!
- resonable near

- too small near
$\rightarrow$ z-values very close
$\rightarrow$ depth order can get lost



## Normalizing Transformation

- $Z \rightarrow \frac{n+f}{f-n}-\frac{2 n f}{(f-n) z}$
- If $z$ is between planes n and f , it is mapped to $[-1,1]$
- If $Z$ is between 0 and $n$, i.e. between camera and near plane, it is mapped to the interval $[-\infty,-1]$
- If $z$ is positive, i.e. behind the camera, then it remains positive after mapping.



## Pipeline

- Transformations in a pipeline



## Pipeline

- additionally, we add a model transformation
- this maps the local coordinates of an object to the world



## Viewing Transformation

- several coordinate systems

3D object/model coords


## All together

- eye,at,up:
viewing paramters
= extrinsic parameters
- fovy, aspect, near, far: perspective parameters = intrinsic parameters



## In OpenGL / WebGL

- In old OpenGL versions, matrices where handled by OpenGL:
- there is one matrix PROJECTION
$\rightarrow$ orthogonal projection matrix set by:
glortho(left, right, bottom, top, near, far);
$\rightarrow$ perspective matrix set by
glFrustum(left, right, bottom, top, near, far);
$\rightarrow$ or by
gluPerspective(fovy, aspect, near, far);
- Viewing matrix and model matrix are stored as one MODELVIEW matrix
$\rightarrow$ first, viewing is set using
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
where the view direction is set using a lookat point: $g=a t-e y e$
$\rightarrow$ then modeling transformations can be appended, e.g. using
glTranslate(...), glRotate(...), glMultMatrix(...)
- To every vertex, first the MODELVIEW and then the PROJECTION matrix is applied before rasterization


## In OpenGL / WebGL

- New OpenGL and WebGL have to do all this in the vertex shader
- So the matrix stuff must happen by the application
- In javascript: libraries, e.g. gl-Matrix.js
- and then upload the matrices as uniforms
- Visibility: now that we have depth, how can we compute occlusion ?

