
MARCH 5-9
2007
SAN FRANCISCO
MOSCONE CENTER
TAKE CONTROL
www.gdconf.com

Projection Matrix Tricks

Eric Lengyel

Terathon Software

Outline

- Projection Matrix Internals
- Infinite Projection Matrix
- Depth Modification
- Oblique Near Clipping Plane

- Slides available at http://www.terathon.com/

From Camera to Screen

Camera Space

↓ Projection Matrix

Homogeneous Clip Space

↓ Perspective Divide

Normalized Device Coordinates

↓ Viewport Transform

Viewport Coordinates

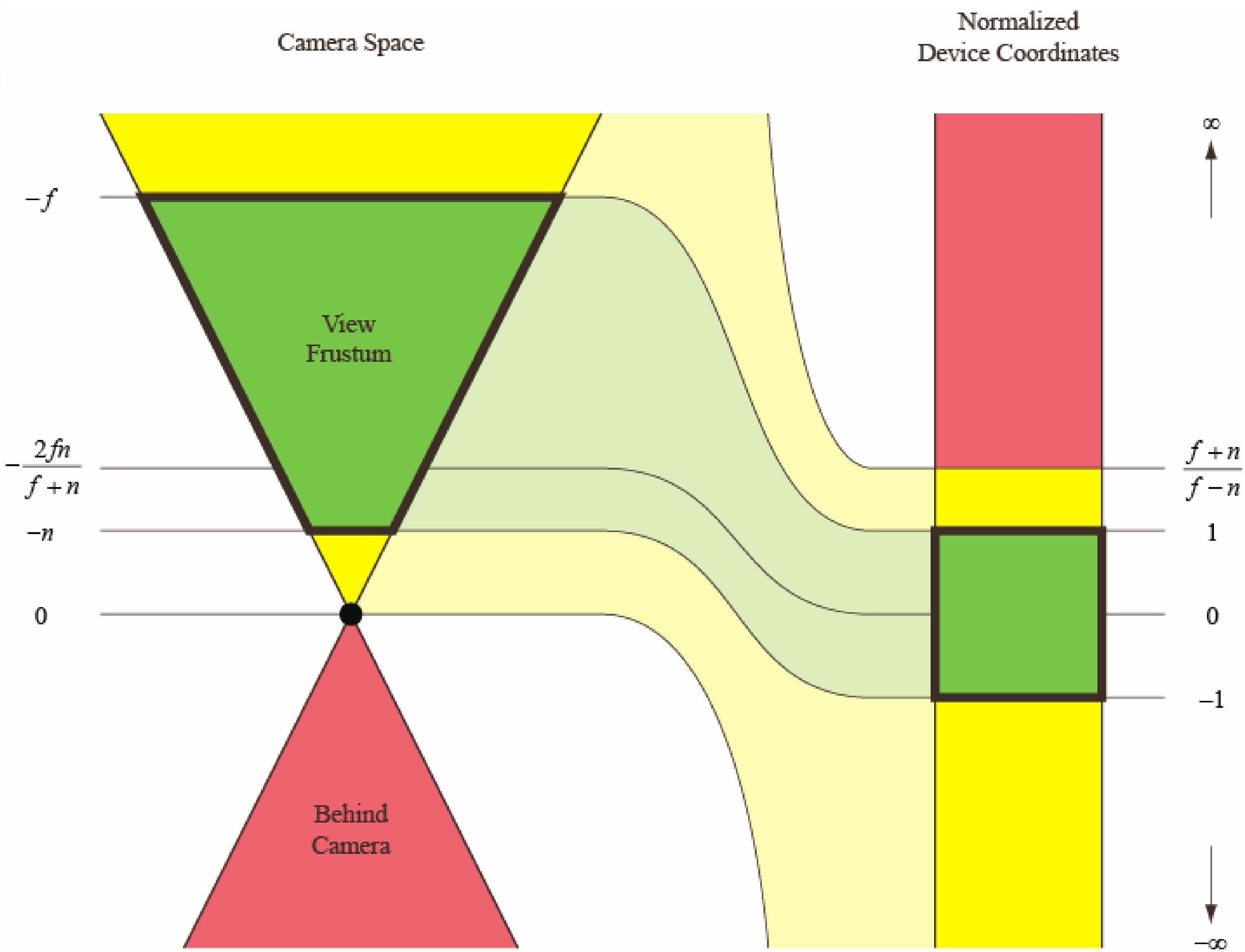
Projection Matrix

- The 4×4 projection matrix is really just a linear transformation in homogeneous space
- It doesn't actually perform the projection, but just sets things up right for the next step
- The projection occurs when you divide by *w* to get from homogenous coordinates to 3-space

OpenGL projection matrix

- n, f = distances to near, far planes
- e = focal length = 1 / tan(FOV / 2)
- a = viewport height / width

$$\begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & -\dfrac{f+n}{f-n} & -\dfrac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$


Camera Space
Normalized Device Coordinates
$-f$
View Frustum
$-\frac{2fn}{f+n}$
$-n$
0
Behind Camera
∞
$\frac{f+n}{f-n}$
1
0
-1
$-\infty$

Infinite Projection Matrix

- Take limit as *f* goes to infinity

$$\lim_{f \to \infty} \begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & -\dfrac{f+n}{f-n} & -\dfrac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Infinite Projection Matrix

- Directions are mapped to points on the infinitely distant far plane
- A direction is a 4D vector with $w = 0$ (and at least one nonzero x, y, z)
- Good for rendering sky objects
  - Skybox, sun, moon, stars
- Also good for rendering stencil shadow volume caps

Infinite Projection Matrix

- The important fact is that *z* and *w* are equal after transformation to clip space:

$$\begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} ex \\ (e/a)y \\ -z \\ -z \end{bmatrix}$$

Infinite Projection Matrix

- After perspective divide, the z coordinate should be exactly 1.0, meaning that the projected point is precisely on the far plane:

$$\begin{bmatrix} ex \\ (e/a)y \\ -z \\ -z \end{bmatrix} \rightarrow \begin{bmatrix} -ex/z \\ -ey/az \\ 1 \end{bmatrix}$$

Infinite Projection Matrix

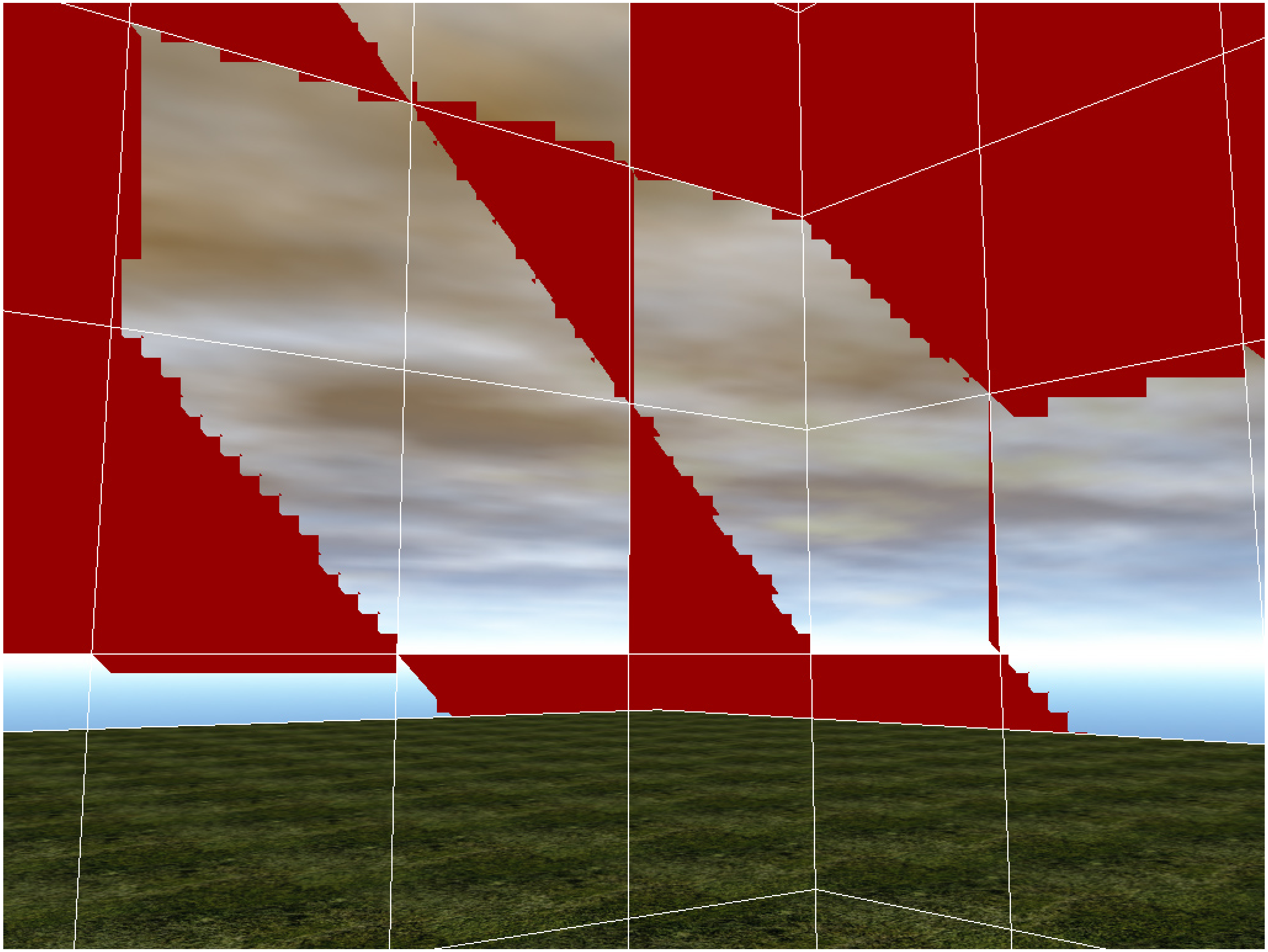
- But there's a problem...
- The hardware doesn't actually perform the perspective divide immediately after applying the projection matrix
- Instead, the viewport transformation is applied to the (x, y, z) coordinates first

Infinite Projection Matrix

- Ordinarily, z is mapped from the range $[-1, 1]$ in NDC to $[0, 1]$ in viewport space by multiplying by 0.5 and adding 0.5
- This operation can result in a loss of precision in the lowest bits
- Result is a depth slightly smaller than 1.0 or slightly bigger than 1.0

Infinite Projection Matrix

- If the viewport-space *z* coordinate is slightly bigger than 1.0, then fragment culling occurs
- The hardware thinks the fragments are beyond the far plane
- Can be corrected by enabling GL_DEPTH_CLAMP_NV, but this is a vendor-specific solution



Infinite Projection Matrix

- Universal solution is to modify projection matrix so that viewport-space *z* is always slightly less than 1.0 for points on the far plane:

$$\begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ 0 & 0 & \varepsilon - 1 & (\varepsilon - 2)n \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Infinite Projection Matrix

- This matrix still maps the near plane to −1, but the infinite far plane is now mapped to 1 − ε

$$\begin{bmatrix} \varepsilon - 1 & (\varepsilon - 2)n \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -n \\ 1 \end{bmatrix} = \begin{bmatrix} -n \\ n \end{bmatrix}$$

$$\begin{bmatrix} \varepsilon - 1 & (\varepsilon - 2)n \\ -1 & 0 \end{bmatrix} \begin{bmatrix} z \\ 0 \end{bmatrix} = \begin{bmatrix} z(\varepsilon - 1) \\ -z \end{bmatrix}$$

Depth Modification

- Several methods exist for performing polygon offset
  - Hardware support through glPolygonOffset
  - Fiddle with glDepthRange
  - Tweak the projection matrix

Depth Modification

- glPolygonOffset works well, but
  - Can adversely affect hierarchical *z* culling performance
  - Not guaranteed to be consistent across different GPUs
- Adjusting depth range
  - Reduces overall depth precision
- Both require extra state changes

Depth Modification

- NDC depth is given by a function of the lower-right 2×2 portion of the projection matrix:

$$\begin{bmatrix} -\dfrac{f+n}{f-n} & -\dfrac{2fn}{f-n} \\ -1 & 0 \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix} = \begin{bmatrix} -\dfrac{f+n}{f-n} z - \dfrac{2fn}{f-n} \\ -z \end{bmatrix}$$

$$z_{NDC} = \frac{f+n}{f-n} + \frac{2fn}{z(f-n)}$$

Depth Modification

- We can add a constant offset ε to the NDC depth as follows:

$$\begin{bmatrix} -\dfrac{f+n}{f-n}-\varepsilon & -\dfrac{2fn}{f-n} \\ -1 & 0 \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix} = \begin{bmatrix} \left(-\dfrac{f+n}{f-n}-\varepsilon\right) z - \dfrac{2fn}{f-n} \\ -z \end{bmatrix}$$

$$z_{NDC} = \frac{f+n}{f-n} + \frac{2fn}{z(f-n)} + \varepsilon$$

Depth Modification

- *w*-coordinate unaffected
- Thus, *x* and *y* coordinates unaffected
- *z* offset is constant in NDC
- But this is not constant in camera space
- For a given offset in camera space, the corresponding offset in NDC depends on the depth

Depth Modification

- What happens to a camera-space offset δ?

$$\begin{bmatrix} -\dfrac{f+n}{f-n} & -\dfrac{2fn}{f-n} \\ -1 & 0 \end{bmatrix} \begin{bmatrix} z+\delta \\ 1 \end{bmatrix} = \begin{bmatrix} -\dfrac{f+n}{f-n}(z+\delta) - \dfrac{2fn}{f-n} \\ -(z+\delta) \end{bmatrix}$$

$$z_{NDC} = \frac{f+n}{f-n} + \frac{2fn}{z(f-n)} - \frac{2fn}{f-n}\left(\frac{\delta}{z(z+\delta)}\right)$$

Depth Modification

- NDC offset as a function of camera-space offset δ and camera-space *z*:

$$\varepsilon(\delta, z) = -\frac{2fn}{f-n}\left(\frac{\delta}{z(z+\delta)}\right)$$

- Remember, δ is negative for an offset toward camera

Depth Modification

- Need to make sure that ε is big enough to make a difference in a typical 24-bit integer *z* buffer
- NDC range of $[-1,1]$ is divided into 2^{24} possible depth values
- So $|\varepsilon|$ should be at least $2/2^{24} = 2^{-23}$

Depth Modification

- But we're adding ε to $(f + n)/(f - n)$, which is close to 1.0
- Not enough bits of precision in 32-bit float for this
- So in practice, it's necessary to use

$$|\varepsilon| \geq 2^{-21} \approx 4.77 \times 10^{-7}$$

Oblique Near Clipping Plane

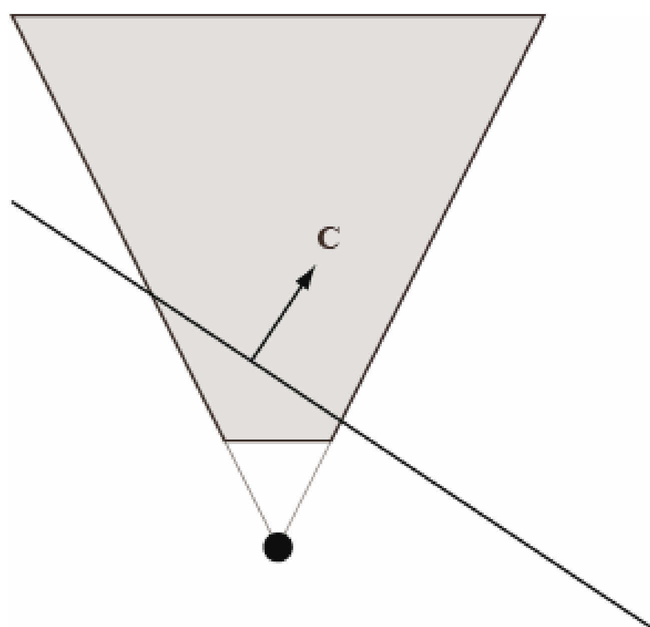
- It’s sometimes necessary to restrict rendering to one side of some arbitrary plane in a scene
- For example, mirrors and water surfaces
- Using an extra hardware clipping plane seems like the ideal solution

Oblique Near Clipping Plane

- But some older hardware doesn't support user clipping planes
- Enabling a user clipping plane could require modifying your vertex programs
- There's a slight chance that a user clipping plane will slow down your fragment programs

Oblique Near Clipping Plane

- Extra clipping plane almost always redundant with near plane
- No need to clip against both planes

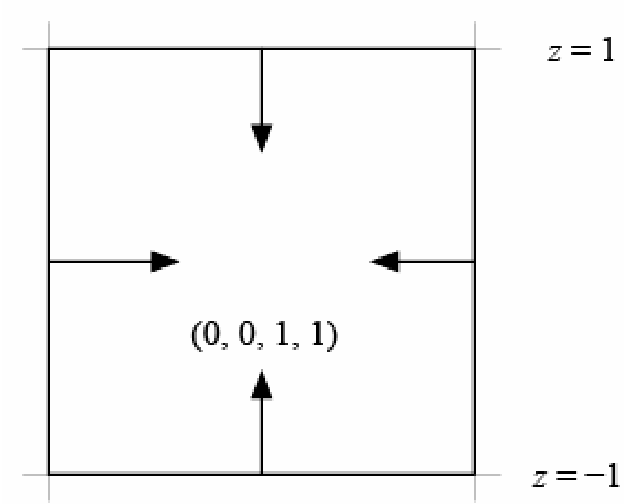


Oblique Near Clipping Plane

- We can modify the projection matrix so that the near plane is moved to an arbitrary location
- No user clipping plane required
- No redundancy

Oblique Near Clipping Plane

- In NDC, the near plane has coordinates (0, 0, 1, 1)



Oblique Near Clipping Plane

- Planes are transformed from NDC to camera space by the transpose of the projection matrix
- So the plane (0, 0, 1, 1) becomes $M_3 + M_4$, where M_i is the i-th row of the projection matrix
- M_4 must remain (0, 0, −1, 0) so that perspective correction still works right

Oblique Near Clipping Plane

- Let C = (C_x, C_y, C_z, C_w) be the camera-space plane that we want to clip against instead of the conventional near plane
- We assume the camera is on the negative side of the plane, so $C_w < 0$
- We must have C = M_3 + M_4, where M_4 = (0, 0, −1, 0)

Oblique Near Clipping Plane

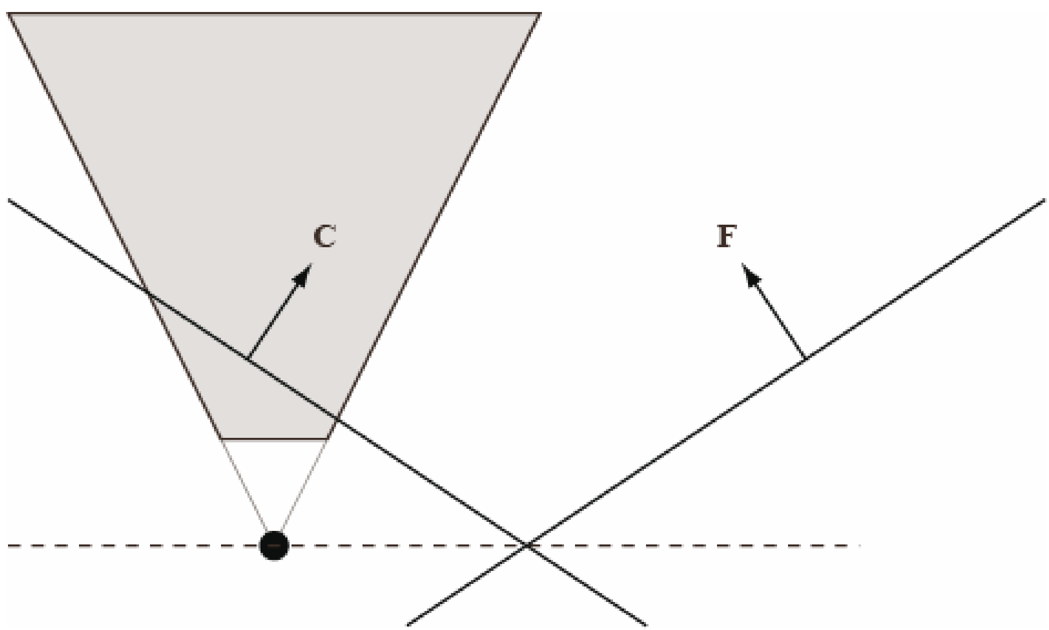
- $M_3 = C - M_4 = (C_x, C_y, C_z + 1, C_w)$

$$\mathbf{M} = \begin{bmatrix} e & 0 & 0 & 0 \\ 0 & e/a & 0 & 0 \\ C_x & C_y & C_z + 1 & C_w \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- This matrix maps points on the plane C to the $z = -1$ plane in NDC

Oblique Near Clipping Plane

- But what happens to the far plane?
- $F = M_4 - M_3 = 2M_4 - C$



Oblique Near Clipping Plane

- Far plane is completely hosed!
- Depths in NDC no longer represent distance from camera plane, but correspond to the position between the oblique near and far planes
- We can minimize the effect, and in practice it’s not so bad

Oblique Near Clipping Plane

- We still have a free parameter:
  the clipping plane C can be scaled
- Scaling C has the effect of changing the orientation of the far plane F
- We want to make the new view frustum as small as possible while still including the conventional view frustum

Oblique Near Clipping Plane

- Let F = $2M_4 - a\mathrm{C}$
- Choose the point Q which lies opposite the near plane in NDC:

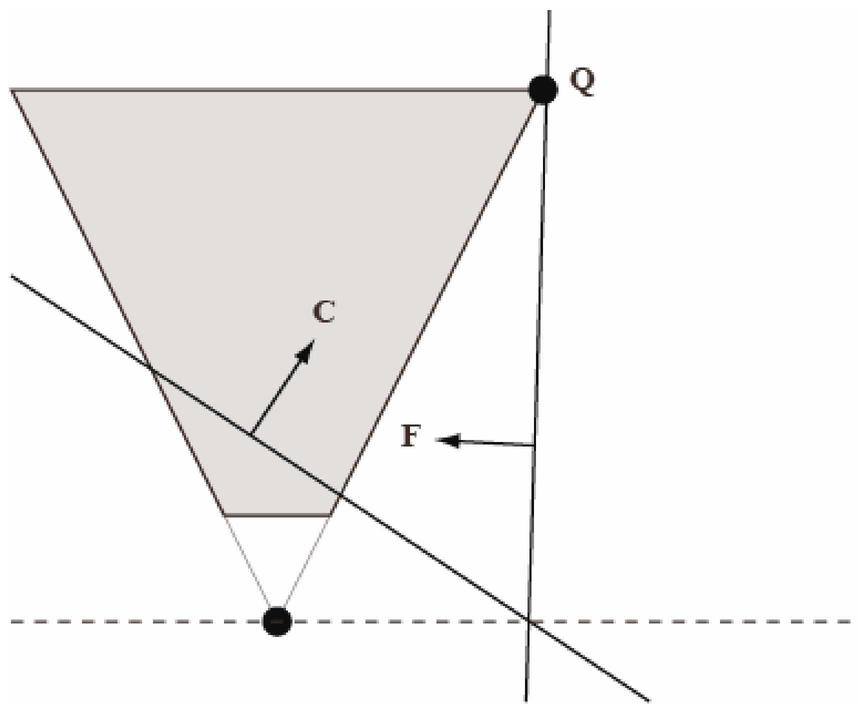
$$\mathbf{Q} = \left(\operatorname{sgn}(C_x), \operatorname{sgn}(C_y), 1, 1\right)$$

- Solve for *a* such that Q lies in plane F:

$$a = \frac{2\mathbf{M}_4 \cdot \mathbf{Q}}{\mathbf{C} \cdot \mathbf{Q}}$$

Oblique Near Clipping Plane

- Near plane doesn't move, but far plane becomes optimal



Oblique Near Clipping Plane

- This also works for infinite view frustum
- Far plane ends up being parallel to one of the edges between two side planes

- For more analysis, see *Journal of Game Development*, Vol 1, No 2

Questions?

- lengyel@terathon.com