Projective texture mapping. Shadow maps

Mapping from desired to reference image

$$\dot{C}_1 + (\bar{c}_1 + u_1\bar{a}_1 + v_1\bar{b}_1)w_1 = \dot{C}_2 + (\bar{c}_2 + u_2\bar{a}_2 + v_2\bar{b}_2)w_2$$

$$\begin{bmatrix}\bar{a}_2 & \bar{b}_2 & \bar{c}_2\end{bmatrix}\begin{bmatrix}w_2u_2\\ w_2v_2\\ w_2\end{bmatrix} = \dot{C}_1 - \dot{C}_2 + \begin{bmatrix}\bar{a}_1 & \bar{b}_1 & \bar{c}_1\end{bmatrix}\begin{bmatrix}w_1u_1\\ w_1v_1\\ w_1\end{bmatrix}$$

$$\begin{bmatrix}w_2u_2\\ w_2v_2\\ w_2\end{bmatrix} = \begin{bmatrix}\bar{a}_2 & \bar{b}_2 & \bar{c}_2\end{bmatrix}^{-1}(\dot{C}_1 - \dot{C}_2) + \begin{bmatrix}\bar{a}_2 & \bar{b}_2 & \bar{c}_2\end{bmatrix}^{-1}\begin{bmatrix}\bar{a}_1 & \bar{b}_1 & \bar{c}_1\end{bmatrix}\begin{bmatrix}w_1u_1\\ w_1v_1\\ w_1\end{bmatrix}$$

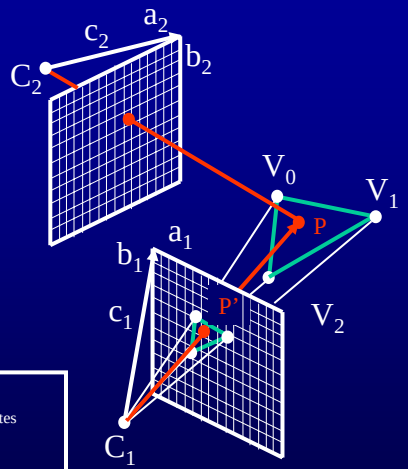
$$\begin{bmatrix}w_2u_2\\ w_2v_2\\ w_2\end{bmatrix} = \begin{bmatrix}q_{00}\\ q_{10}\\ q_{20}\end{bmatrix} + \begin{bmatrix}q_{01} & q_{02} & q_{03}\\ q_{11} & q_{12} & q_{13}\\ q_{21} & q_{22} & q_{23}\end{bmatrix}\begin{bmatrix}w_1u_1\\ w_1v_1\\ w_1\end{bmatrix}$$

$$u_2 = \frac{\frac{q_{00}}{w_1} + q_{01}u_1 + q_{02}v_1 + q_{03}}{\frac{q_{20}}{w_1} + q_{21}u_1 + q_{22}v_1 + q_{23}}$$

$$\frac{1}{w_1} = Au_1 + Bv_1 + C$$

$$u_2 = \frac{Du_1 + Ev_1 + F}{Ju_1 + Kv_1 + L}$$

$$v_2 = \frac{Gu_1 + Hv_1 + I}{Ju_1 + Kv_1 + L}$$



- (C_2, a_2, b_2, c_2) – reference view
- (u_2, v_2) – reference pixel coordinates
- (C_1, a_1, b_1, c_1) – desired view
- (u_1, v_1) – desired view pixel coordinates
- (u_2, v_2, w_2) – unknowns
- w_1 – computed using triangle $V_0V_1V_2$

Shadow Maps

- Efficient implementation of shadows
- Essentially a zbuffer rendered from the light
  - if a point is behind the map as seen from the light, it is in shadow
  - the z-values model the first-surfaces seen from the light

Shadow Map Implementation

- Step 1: construction
  - one per light
  - updated when light or objects move
  - does not need to be updated when only the camera moves
  - resolution according to
    - scene geometry
    - desired image resolution
    - desired shadow quality
    - budget
  - view should
    - cover all light rays
    - cover all scene (cube maps if needed)
    - near / far plane according to scene bounding box

Shadow Map Implementation

- Step 2: shadow computation
  - project scene point visible at current pixel onto shadow map(s)
  - if hidden, pixel is in shadow
  - else light contributes to pixel color
  - soft shadows
    - pixels close to the shadow border are partially in shadow (penumbra)
    - implemented by testing neighborhood in the shadow map
      - if all samples of neighborhood are in shadow -> shadow
      - if all samples of neighborhood are in the light -> light
      - if *k* of *n* samples are in the light -> penumbra (light contribution *k*/*n*)