Volumetric Image Visualization

Alexandre Xavier Falcão

LIDS - Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

From 3D to 2D

This lecture is organized as follows.

From 3D to 2D

This lecture is organized as follows.

- Rendering — a process that creates an image (rendition) in a viewing plane with some object information from a 3D image (scene).

From 3D to 2D

This lecture is organized as follows.

- Rendering — a process that creates an image (rendition) in a viewing plane with some object information from a 3D image (scene).
- Rendering with orthogonal projection — a geometric transformation that avoids distortions in the interpretation of the scene.

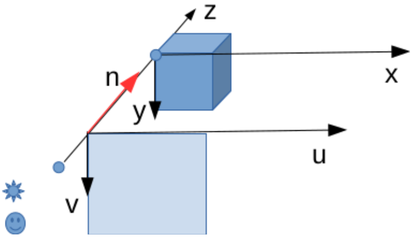
From 3D to 2D

This lecture is organized as follows.

- Rendering — a process that creates an image (rendition) in a viewing plane with some object information from a 3D image (scene).
- Rendering with orthogonal projection — a geometric transformation that avoids distortions in the interpretation of the scene.
- A simple rendering example — how to draw the wireframe of a scene.

Rendering

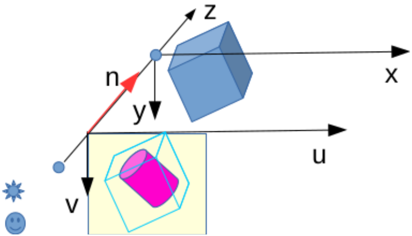
By using some illumination model, one can simulate the transformation of a scene followed by the projection onto a viewing plane *uv* of the light reflected on the surface of each object inside the scene.



An observer and a white light source may be assumed to be at an infinite distance ($z = -\infty$) to the viewing plane. One can also simulate the inverse transformation of the observer, light source, and viewing plane.

Rendering

By using some illumination model, one can simulate the transformation of a scene followed by the projection onto a viewing plane uv of the light reflected on the surface of each object inside the scene.



An observer and a white light source may be assumed to be at an infinite distance ($z = -\infty$) to the viewing plane. One can also simulate the inverse transformation of the observer, light source, and viewing plane.

Orthogonal projection

- A projection is a change from the scene coordinate system (x, y, z) to the coordinate system (u, v, n) of the viewing plane, where $\mathrm{n} = (0, 0, 1, 0)$ is the viewing direction and $-\mathrm{v} = (0, -1, 0, 0)$ is the view-up vector.

Orthogonal projection

- A projection is a change from the scene coordinate system (x, y, z) to the coordinate system (u, v, n) of the viewing plane, where $\mathrm{n} = (0, 0, 1, 0)$ is the viewing direction and $-\mathrm{v} = (0, -1, 0, 0)$ is the view-up vector.
- By rotating a scene, with diagonal d, around its center $c = (x_c, y_c, z_c, 1)$ and axes x (*tilt*) and y (*spin*), one obtains its orthogonal projection in uv as

$$
\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \mathrm{T}(\frac{d}{2}, \frac{d}{2}, \frac{d}{2})\mathrm{R}_y(\beta)\mathrm{R}_x(\alpha)\mathrm{T}(-x_c, -y_c, -z_c) \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}
$$

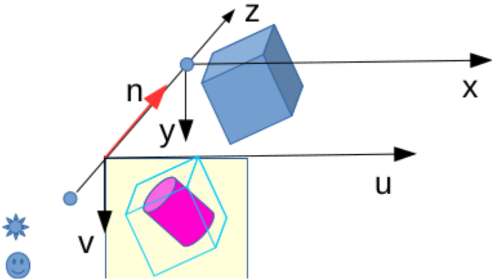
$$
u = x_2
$$

$$
v = y_2
$$

where changes in $\mathrm{T}(\frac{d}{2}, \frac{d}{2}, \frac{d}{2})$ and on the position of uv might cause clipping and cut of the scene.

Orthogonal projection

The value z_2 represents the distance between xy and the object's surface, so those distance values may be stored in a z-buffer and used for hidden surface removal or, together with the illumination model, for showing the closer objects brighter in the rendition $\hat{J} = (D_J, J)$, with $d \times d$ pixels to avoid clipping.



The bounding box of the scene (cyan) is called its wireframe.

Drawing the wireframe of the scene

- Consider the problem of drawing the visible edges of the scene's wireframe in uv, for different tilt and spin angles, by assuming the faces of the scene are opaque.

Drawing the wireframe of the scene

- Consider the problem of drawing the visible edges of the scene's wireframe in uv, for different tilt and spin angles, by assuming the faces of the scene are opaque.

- As strategy, one can

  - first determine which faces are visible after transformation and,

  - for each edge, if both of its extreme points belong to a visible face,

    - they must be projected and

    - a line must be drawn between them in the uv plane.

Graphical context

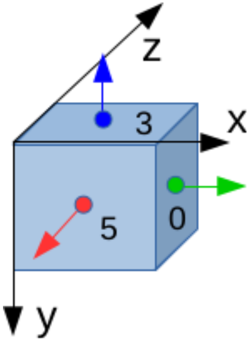
A first step in rendering is to define the graphical context, which must store all information needed for the rendering.

- The transformations that will be applied to points and vectors, respectively.

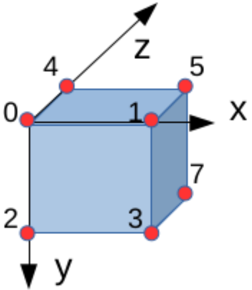
$$
\begin{aligned}
\Phi &= \mathrm{T}(\frac{d}{2},\frac{d}{2},\frac{d}{2})\mathrm{R}_y(\beta)\mathrm{R}_x(\alpha)\mathrm{T}(-x_c,-y_c,-z_c) \\
\Phi_r &= \mathrm{R}_y(\beta)\mathrm{R}_x(\alpha)
\end{aligned}
$$

Graphical context

- A set $\mathcal{F}$ with the faces of the scene, as represented by their centers and normal vectors, respectively:
  0: $(n_x - 1, \frac{n_y}{2}, \frac{n_z}{2}, 1)$ and $(1, 0, 0, 0)$; 1: $(0, \frac{n_y}{2}, \frac{n_z}{2}, 1)$ and $(-1, 0, 0, 0)$; 2: $(\frac{n_x}{2}, n_y - 1, \frac{n_z}{2}, 1)$ and $(0, 1, 0, 0)$;
  3: $(\frac{n_x}{2}, 0, \frac{n_z}{2}, 1)$ and $(0, -1, 0, 0)$; 4: $(\frac{n_x}{2}, \frac{n_y}{2}, n_z - 1, 1)$ and $(0, 0, 1, 0)$; and 5: $(\frac{n_x}{2}, \frac{n_y}{2}, 0, 1)$ and $(0, 0, -1, 0)$.



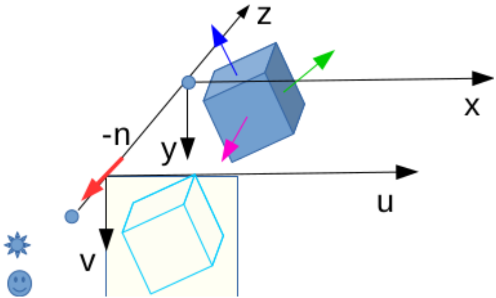
- A set $\mathcal{V}$ with the vertices of the scene:
  $0: (0, 0, 0, 1)$; $1: (n_x - 1, 0, 0, 1)$; $2: (0, n_y - 1, 0, 1)$;
  $3: (n_x - 1, n_y - 1, 0, 1)$; $4: (0, 0, n_z - 1, 1)$;
  $5: (n_x - 1, 0, n_z - 1, 1)$; $6: (0, n_y - 1, n_z - 1, 1)$;
  $7: (n_x - 1, n_y - 1, n_z - 1, 1)$;



- A set $\mathcal{E}$ with the edges of the scene by indicating which vertices compose each edge (e.g., edge 0 connects vertices 0 and 1).

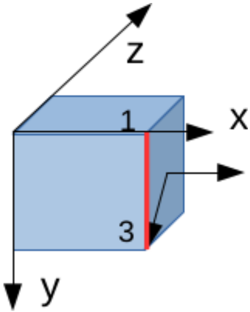
A visible face

A face $f \in \mathcal{F}$ is visible if, after transformation $\Phi_r(f.\text{n})$ of its normal vector $f.\text{n}$, the inner product $\langle \Phi_r(f.\text{n}), -\text{n} \rangle > 0$, where $-\text{n} = (0, 0, -1, 0)$.



A point in a face

The extreme points $e.p_1$ and $e.p_n$ of an edge $e \in \mathcal{E}$ fall in a face $f \in \mathcal{F}$ with center at $f.c$, iff $\langle e.p_1 - f.c, f.\mathrm{n} \rangle = 0$ and $\langle e.p_n - f.c, f.\mathrm{n} \rangle = 0$.



Drawing an edge of a visible face

- By identifying that a face $f \in \mathcal{F}$ is visible and an edge $e \in \mathcal{E}$ is part of that face, we must transform the points of e, by $p_1 \leftarrow \Phi(e.p_1)$ and $p_n \leftarrow \Phi(e.p_n)$, and draw a line in uv (i.e., image $\hat{J} = (D_J, J)$) from $p_1 = (u_{p_1}, v_{p_1})$ to $p_n = (u_{p_n}, v_{p_n})$.

Drawing an edge of a visible face

- By identifying that a face $f \in \mathcal{F}$ is visible and an edge $e \in \mathcal{E}$ is part of that face, we must transform the points of e, by $p_1 \leftarrow \Phi(e.p_1)$ and $p_n \leftarrow \Phi(e.p_n)$, and draw a line in uv (i.e., image $\hat{J} = (D_J, J)$) from $p_1 = (u_{p_1}, v_{p_1})$ to $p_n = (u_{p_n}, v_{p_n})$.
- This line can be drawn by using the *Digital Differential Analyzer* (DDA) algorithm.

The DDA algorithm in 2D

- Let $p_k = (u_{p_k}, v_{p_k})$, $k = 1, 2, \ldots, n$, be the n points drawn from p_1 to p_n in uv. Each subsequent point $p_{k+1} = (u_{p_{k+1}}, v_{p_{k+1}})$ is obtained by

$$(u_{p_{k+1}}, v_{p_{k+1}}) = (u_{p_k}, v_{p_k}) + (d_u, d_v),$$

where the displacement (d_u, d_v) follows the direction and orientation of the line segment from p_1 to p_n. These points must also be approximated to pixels in D_J.

The DDA algorithm in 2D

- Let $p_k = (u_{p_k}, v_{p_k})$, $k = 1, 2, \ldots, n$, be the n points drawn from p_1 to p_n in uv. Each subsequent point $p_{k+1} = (u_{p_{k+1}}, v_{p_{k+1}})$ is obtained by

$$(u_{p_{k+1}}, v_{p_{k+1}}) = (u_{p_k}, v_{p_k}) + (d_u, d_v),$$

  where the displacement (d_u, d_v) follows the direction and orientation of the line segment from p_1 to p_n. These points must also be approximated to pixels in D_J.

- In order to avoid that a pixel be visited multiple times, d_u and d_v can be computed as $d_a = sign(a_{p_n} - a_{p_1}) \in \{-1, 1\}$, $a \in \{u, v\}$.

The DDA algorithm in 2D

- Let $p_k = (u_{p_k}, v_{p_k})$, $k = 1, 2, \ldots, n$, be the n points drawn from p_1 to p_n in uv. Each subsequent point $p_{k+1} = (u_{p_{k+1}}, v_{p_{k+1}})$ is obtained by

$$(u_{p_{k+1}}, v_{p_{k+1}}) = (u_{p_k}, v_{p_k}) + (d_u, d_v),$$

  where the displacement (d_u, d_v) follows the direction and orientation of the line segment from p_1 to p_n. These points must also be approximated to pixels in D_J.

- In order to avoid that a pixel be visited multiple times, d_u and d_v can be computed as $d_a = sign(a_{p_n} - a_{p_1}) \in \{-1, 1\}$, $a \in \{u, v\}$.

- The maximum intensity H can be used to draw the lines in the output image $\hat{J} = (D_J, J)$.

The DDA algorithm in 2D

Input: Points p_1 and p_n, and intensity H.
Output: Image $\hat{J} = (D_J, J)$.

```
If $p_1 = p_n$ then set $n \leftarrow 1$.
Else
    set $D_u \leftarrow u_{p_n} - u_{p_1}$ and $D_v \leftarrow v_{p_n} - v_{p_1}$.
    If $|D_u| \geq |D_v|$ then
        set $n \leftarrow |D_u| + 1$, $d_u \leftarrow sign(D_u)$, and $d_v \leftarrow \frac{d_u D_v}{D_u}$.
    Else
        set $n \leftarrow |D_v| + 1$, $d_v \leftarrow sign(D_v)$, and $d_u \leftarrow \frac{d_v D_u}{D_v}$.
Set $p = (u_p, v_p) \leftarrow (u_{p_1}, v_{p_1})$.
For each $k = 1$ to $n$, do
    Set $J(p) \leftarrow H$.
    Set $p = (u_p, v_p) \leftarrow (u_p, v_p) + (d_u, d_v)$.
```

Adding attributes to the rendering

- The lines may be drawn with thickness $2r$ by changing $J(p) \leftarrow H$ to $\forall q \in \mathcal{A}_r(p), J(q) \leftarrow H$, where $q \in \mathcal{A}_r(p)$ if $1 \leq \|q - p\| \leq r$.

Adding attributes to the rendering

- The lines may be drawn with thickness $2r$ by changing $J(p) \leftarrow H$ to $\forall q \in \mathcal{A}_r(p), J(q) \leftarrow H$, where $q \in \mathcal{A}_r(p)$ if $1 \leq \|q - p\| \leq r$.

- One can also draw colored lines by painting their red, green, and blue values in three bands of a colored image $\hat{J} = (D_J, \text{J})$.

Adding attributes to the rendering

- The lines may be drawn with thickness $2r$ by changing $J(p) \leftarrow H$ to $\forall q \in \mathcal{A}_r(p), J(q) \leftarrow H$, where $q \in \mathcal{A}_r(p)$ if $1 \leq \|q - p\| \leq r$.

- One can also draw colored lines by painting their red, green, and blue values in three bands of a colored image $\hat{J} = (D_J, \mathrm{J})$.

- Let's implement this simple algorithm as exercise?

Algorithm to draw a wireframe

Input: Scene $\hat{I} = (D_I, I)$, angles α, and β.
Output: Rendition $\hat{J} = (D_J, J)$ of the wireframe.

1. create a graphical context with $\mathcal{F}$, $\mathcal{E}$, H, Φ, and Φ_r from D_I, α, and β.
2. for each $f \in \mathcal{F}$ do
3.   if $\langle \Phi_r(f.\mathrm{n}), (0, 0, -1, 0) \rangle > 0$ then
4.   for each edge $e \in \mathcal{E}$ do
5.   if $\langle e.p_1 - f.c, f.\mathrm{n} \rangle = 0$ and $\langle e.p_n - f.c, f.\mathrm{n} \rangle = 0$ then
6. $p_1 \leftarrow \Phi(e.p_1)$ and $p_n \leftarrow \Phi(e.p_n)$
7.   DrawLine2D($\hat{J}, p_1, p_n, H$)