

Notes for GAMES101

by xiao-h

Lecture 01: Overview of Computer Graphics

1. What is Computer Graphics?

The use of computers to synthesize and manipulate visual information.

2. Why Study Computer Graphics?

Applications:

1. Video Games
2. Movies
3. Animations
4. Design
5. Visualization
6. Virtual Reality
7. Digital Illustration
8. Simulation
9. Graphical User Interfaces (GUI)
10. Typography

3. Course Topics

- Rasterization
- Curves and Meshes
- Ray Tracing
- Animation / Simulation

Lecture 02: Review of Linear Algebra

1. Review of Linear Algebra

- Vectors
- Matrices

Lecture 03: Transformation

1. Why Study Transformation

- Modeling
- Viewing

2. 2D Transformations: Rotation, Scale, Shear

Scale Matrix

$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear Matrix

Horizontal shift:

$$\begin{cases} x' = x + ay \\ y' = y \end{cases} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation Matrix

2D rotation:

$$R_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

3. Homogeneous Coordinates

Translation

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Translation is NOT linear transform! So, we can introduce **homogenous coordinates**.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

In homogeneous coordinates, $\begin{bmatrix} x \\ y \\ w \end{bmatrix}$ is the 2D point $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ 1 \end{bmatrix}$, $w \neq 0$

Affine Map = Linear Map + Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Using homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

4. 3D Transforms

Use homogeneous coordinates again:

- 3D point = $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$
- 3D vector = $\begin{bmatrix} x & y & z & 0 \end{bmatrix}^T$

In general, $\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$, $w \neq 0$ is the 3D point $\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{bmatrix}$

Use 4x4 matrices for affine transformations.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Linear Transform would be applied firstly, then Translation.

3D rotation and scaling are similar to 2D, and for Rotation:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\alpha) = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To compose and 3D rotation R_{xyz} , we can use Euler angles.

$$R_{xyz}(\alpha, \beta, \gamma) = R_x(\alpha)R_y(\beta)R_z(\gamma)$$

Rodrigues' Rotation Formula:

$$R(\mathbf{n}, \alpha) = \cos(\alpha)\mathbf{I} + (1 - \cos(\alpha))\mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}}_N$$

(TODO: Learn the proof of it.)

The axis passes through the origin.

Lecture 04: Transformation Cont.

1. Viewing Transformation

Think about how to take a photo:

- Find a good place and arrange people (**Model** transformation)
- Find a good “angle” to put the camera (**View** transformation)
- Cheese! (**Projection** transformation)

These three transformations are referred to as **MVP** transformations.

View / Camera Transformation

Define the camera first:

- Position \vec{e}
- Look-at / gaze direction \hat{g}
- Up direction \hat{t}

We always transform the camera to

- The origin, up at Y, look at -Z.
- And Transform the objects along with the camera.

We can transform the camera by M_{view} .

$$M_{\text{view}} = R_{\text{view}}T_{\text{view}}$$

Translate e to origin:

$$T_{\text{view}} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then rotate g to $-Z$, t to Y , $(g \times t)$ to X , but it's hard to construct the matrix. Consider its inverse rotation: X to $(g \times t)$, Y to T , $-Z$ to g .

$$R_{\text{view}}^{-1} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g} \times \hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g} \times \hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow R_{\text{view}} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & y_{\hat{g} \times \hat{t}} & z_{\hat{g} \times \hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

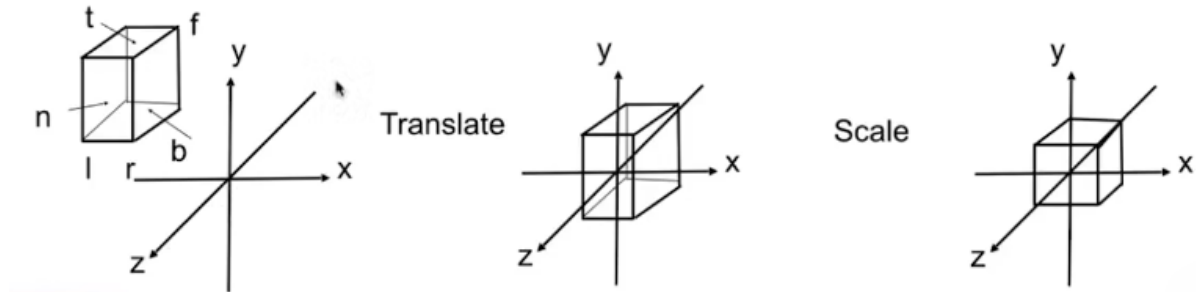
Here R_{view}^{-1} is an **orthogonal matrix** (because the three axes are perpendicular to each other), so its inverse is just its transpose.

Projection Transformation

Orthographic Projection:

Simple way:

- Camera located at origin, looking at $-Z$, up at Y .
- Drop Z coordinate.
- Translate and scale the resulting rectangle to $[-1, 1]^2$.



Real method:

Map a cuboid $[l, r] \times [b, t] \times [f, n]$ to the “canonical” cube $[-1, 1]^3$ (**Attention**, $f < n$).

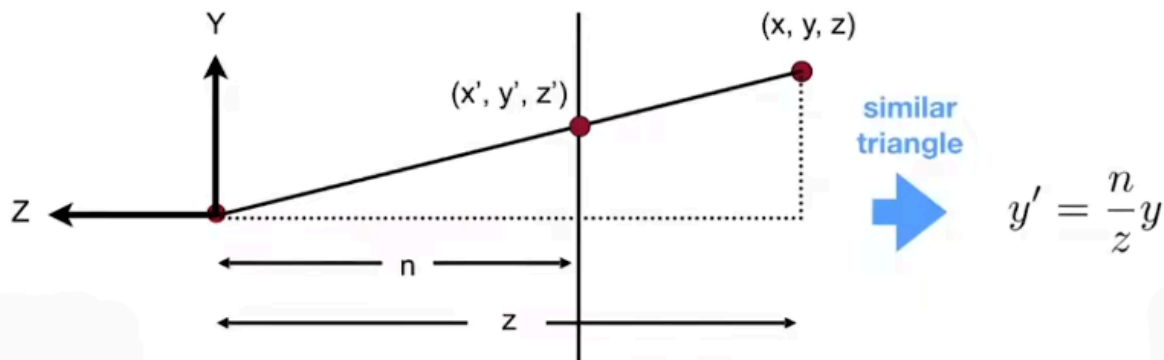
Transformation matrix: translate (center to origin) first, then scale (length/width/height to 2)

$$M_{\text{ortho}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

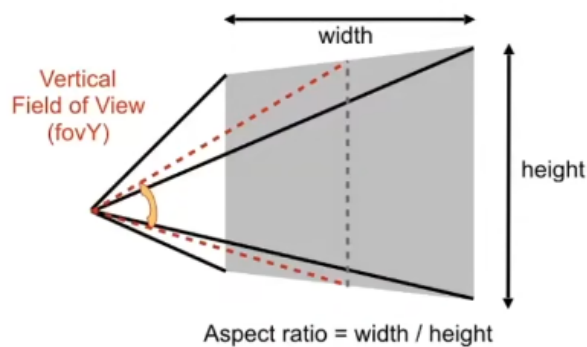
First “squish” the frustum into a cuboid, then do orthographic projection.

$$M_{\text{persp}} = M_{\text{ortho}} M_{\text{persp} \rightarrow \text{ortho}}$$



$$M_{\text{persp} \rightarrow \text{ortho}} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow M_{\text{persp}} = \begin{bmatrix} \frac{2n}{l+r} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{-b+t} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{-f-n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Sometimes people prefer: Vertical field-of-view (fovY) and aspect ratio



$$\tan \frac{\text{fovY}}{2} = \frac{t}{|n|}$$

$$\text{aspect} = \frac{r}{t}$$

Lecture 05: Rasterization 1 (Triangles)

1. Viewport Transform

- Irrelevant to z
- Transform in xy plane: $[-1, 1]^2$ to $[0, \text{width}] \times [0, \text{height}]$

Viewport transform matrix:

$$M_{\text{viewport}} = \begin{bmatrix} \frac{\text{width}}{2} & 0 & 0 & \frac{\text{width}}{2} \\ 0 & \frac{\text{height}}{2} & 0 & \frac{\text{height}}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Triangles - Fundamental Shape Primitives

Why triangles?

- Most basic polygon
- Break up other polygons
- Guaranteed to be planar
- Well-defined interior
- Well-defined method for interpolating values at vertices over triangle

3. Sampling

Sampling: To discretize a function by evaluating at some points.

We can sample if each pixel center is inside triangle:

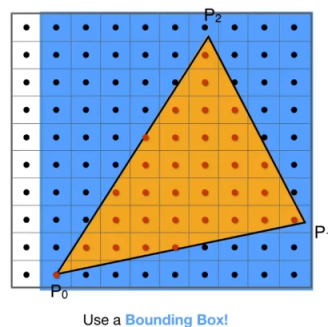
$$\text{inside}(t, x, y) = \begin{cases} 1 & \text{Point } (x, y) \text{ in triangle } t \\ 0 & \text{Otherwise} \end{cases}$$

```
for (int x = 0; x < xmax; ++x)
    for (int y = 0; y < ymax; ++y)
        image[x][y] = inside(tri, x + 0.5, y + 0.5);
```

How to implement $\text{inside}(t, x, y)$? We can use three cross products!

Assume P_1, P_2, P_3 are the three vertices of a triangle, we now want to know whether point Q is inside the triangle. If $\overrightarrow{P_1P_2} \times \overrightarrow{P_1Q}$, $\overrightarrow{P_2P_3} \times \overrightarrow{P_2Q}$ and $\overrightarrow{P_3P_1} \times \overrightarrow{P_3Q}$ have the same sign, we can determine that the point Q is inside the triangle.

We don't need to check all pixels on the screen. We can use a bounding box. The image below shows an **AABB Bounding Box**.



Lecture 06: Rasterization 02 (Antialiasing & Z-Buffering)

1. What is Alias?

Artifacts due to sampling:

- Jaggies
- Moire Patterns
- Wagon Wheel Illusion
- ...

Behind the Aliasing Artifacts: Signals are changing too fast, but sampled too slowly.

Pre-filtering (Blurring) then sampling can do antialiasing.

Fourier Transform: Represent a function as a weighted sum of sines and cosines. (from **spatial domain** to **frequency domain**)

This can transform a function into sines and cosines of different frequencies. But higher frequencies need faster sampling, or the result of the inverse transformation will have a relatively large deviation.

Two frequencies are indistinguishable at a given sampling rate, which is called “**alias**”.

Filter/Convolution/Averaging:

- If we filter out low frequencies (High-pass Filter), there are **only edges left**.
- If we filter out high frequencies (Low-pass Filter), the picture will be **blurred**.

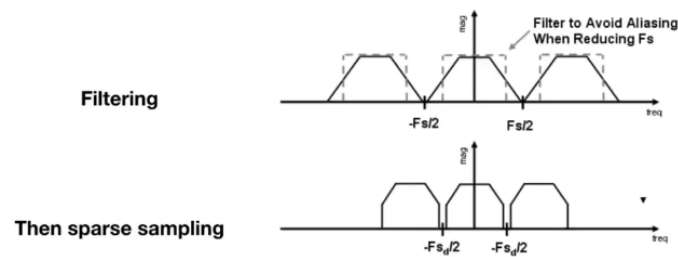
2. Antialiasing

How to reduce alias error?

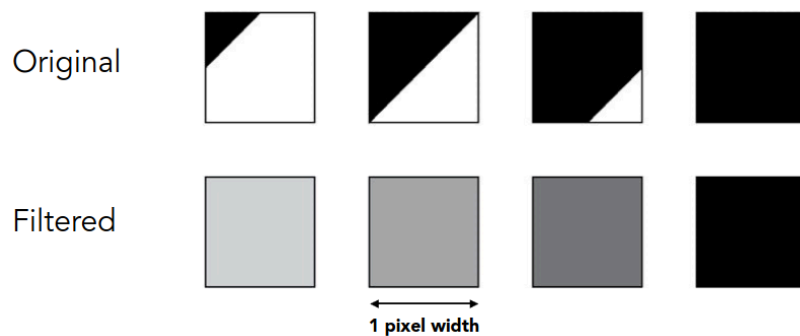
1. Increase sampling rate
2. Antialias

Blur to Antialias

If we blur first and then sample, the high frequencies will be removed, and the spectrum will not overlap easily when it is copied.



A practical pre-filter: Convolve $f(x, y)$ by a 1-pixel box-blur, then sample at every pixel's center.



Supersample to Antialias (MSAA)

Approximate the effect of the 1-pixel box filter by sampling multiple locations within a pixel and averaging their values.

MSAA does not increase the sampling rate, but only approximates reasonable coverage.

In practical applications, the sampling points are not evenly distributed.

Other Methods

- FXAA (Fast Approximate Antialias)
- TAA (Temporal Antialias):

- DLSS (Deep Learning Super Sampling)

3. Z-Buffering

Idea:

- Store current min z-value for each sample (pixel).
- Need an additional buffer for depth values
 - Frame buffer stores color values
 - Depth buffer (z-buffer) stores depth

Lecture 07: Shading 1 (Illumination, Shading & Graphics Pipeline)

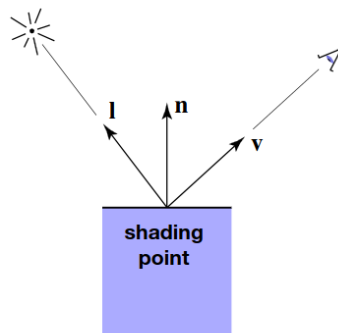
1. What is Shading?

The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

2. Blinn-Phong Reflectance Model

Inputs:

- Viewer direction, v
- Surface normal, n
- Light direction, l
- Surface parameters (color, shininess, ...)

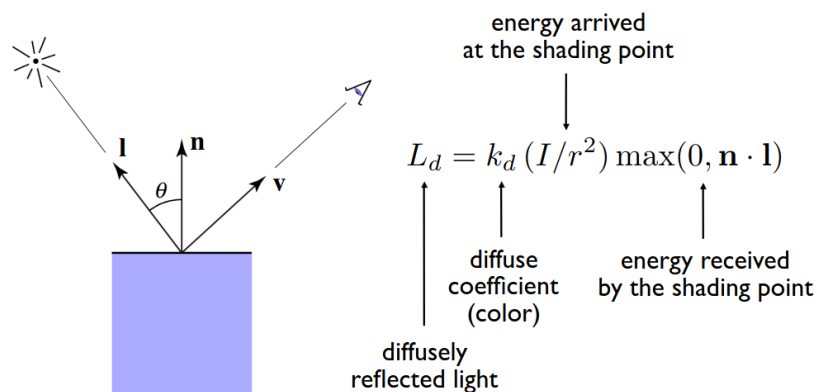


Shading is local, no shadows will be generated (Shading \neq Shadow).

Diffuse Reflection

- Light is scattered uniformly in all directions.

Lambert's cosine law: Light per unit area is proportional to $\cos \theta = l \cdot n$

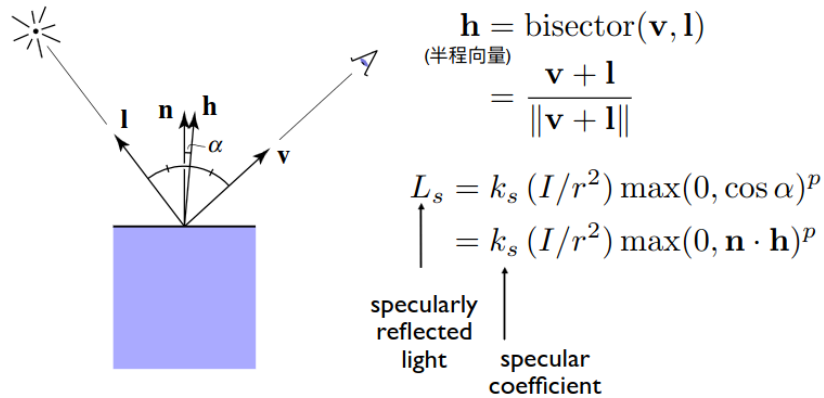


Lecture 08: Shading 2 (Shading, Pipeline & Texture Mapping)

1. Blinn-Phong Reflectance Model Cont.

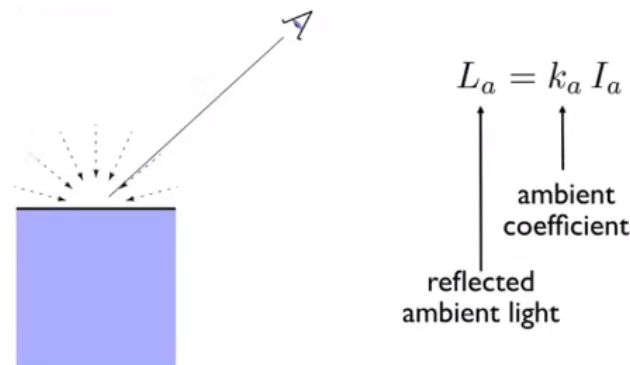
Specular

- Bright near mirror reflection direction.

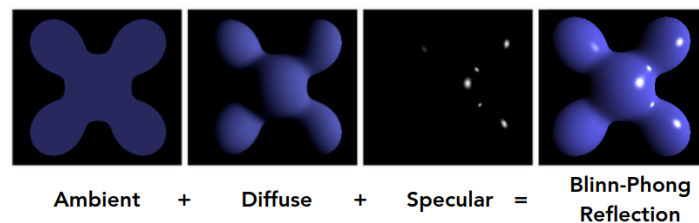


Ambient

- Add constant color to account for disregarded illumination and fill in black shadows.



Blinn-Phong Reflection



$$\begin{aligned}
 L &= L_a + L_d + L_s \\
 &= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p
 \end{aligned}$$

2. Shading Frequencies

Three methods of shading:

- Shade each triangle (**Flat Shading**)

2. Shade each vertex (**Gouraud Shading**)
3. Shade each pixel (**Phong Shading**)

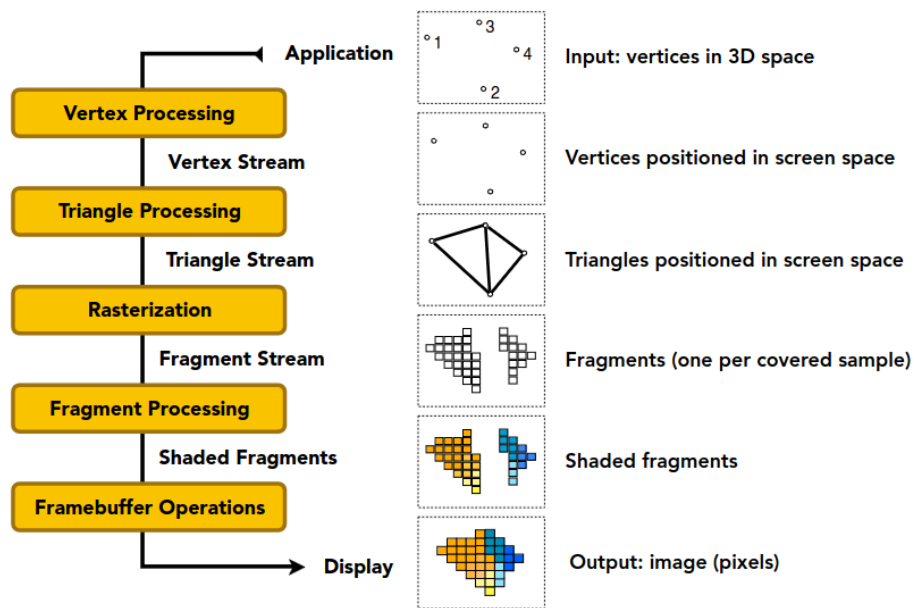
Per-Vertex Normal Vectors:

$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$

Per-Pixel Normal Vectors:

Use **Barycentric Interpolation**:

3. Graphics (Real-Time Rendering) Pipeline



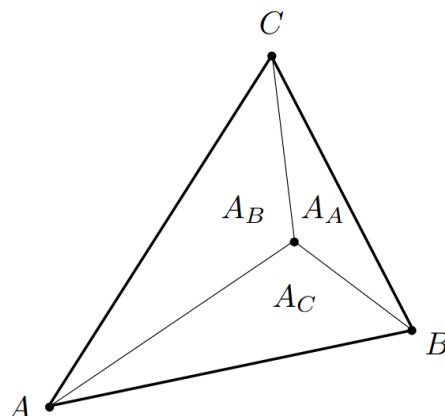
4. Texture Mapping

Texture Space \rightarrow World Space \rightarrow Screen Space The coordinates on the model are mapped to the (u,v) coordinates of the texture.

Lecture 09: Shading 3 (Texture Mapping Cont.)

1. Barycentric Interpolation

Barycentric Coordinates:



$$\alpha = \frac{A_B}{A_A + A_B + A_C}$$

$$\beta = \frac{A_C}{A_A + A_B + A_C}$$

$$\gamma = \frac{A_A}{A_A + A_B + A_C}$$

$$P(x, y) = \alpha A + \beta B + \gamma C, \alpha + \beta + \gamma = 1$$

P is inside the triangle if α, β and γ are all non-negative.

$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

$$V = \alpha V_A + \beta V_B + \gamma V_C$$

V can be texture, coordinates, color, normal, depth, material attributes...

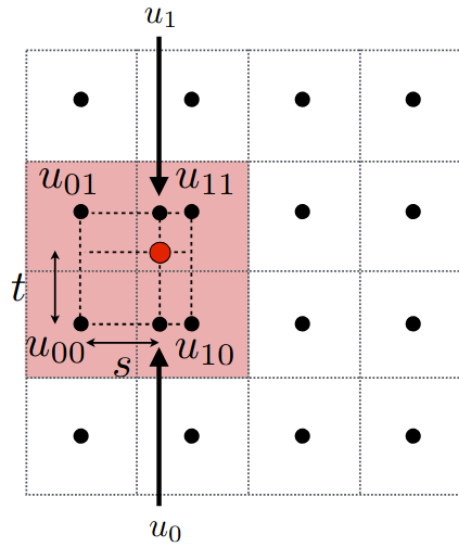
However, barycentric coordinates are **NOT** invariant under projection.

Interpolation should be performed on the original attributes rather than those transformed by projection.

2. Texture Mapping Cont.

Bilinear Interpolation

When the texture resolution is too small, pixels may be mapped to non-integer coordinates on the texture, so we need to interpolate. The simplest way is to round up, but this will result in a grid-like effect. Therefore, we can do interpolation.



Steps:

1. Take 4 nearest sample locations, with texture values as labeled.
2. Get fractional offsets, (s, t) as shown.
3. Define $\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$
4. Two helper lerps: $u_0 = \text{lerp}(s, u_{00}, u_{10})$, $u_1 = \text{lerp}(s, u_{01}, u_{11})$
5. Final lerp: $f(x, y) = \text{lerp}(t, u_0, u_1)$

Going a step further, **Bicubic Interpolation** using the surrounding sixteen points for interpolation can produce smoother results.

Mipmap

If the texture is too large, aliasing will bother us again. That's because signal frequency is too large in a pixel.

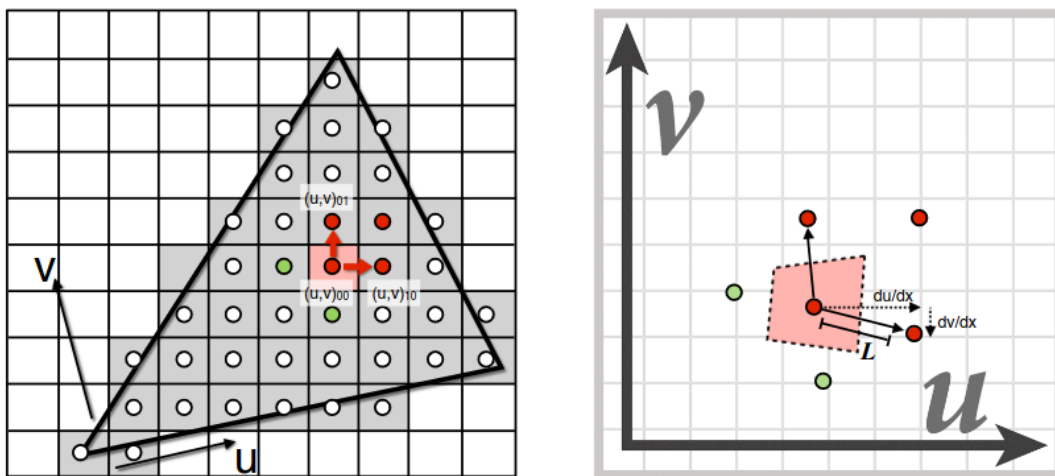
We can just use supersampling, but it's costly. The principle of supersampling is to increase the sampling frequency.

What if we don't sample? We just need to get the average value within a range.

The method is called **Mipmap**. Mipmap allows fast, approximate and square range queries.

Mipmap uses one texture to generate textures of different sizes at different levels. The length and width of each level is half of the previous level.

Computing Mipmap:



$$D = \log_2 L \quad L = \max \left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2} \right)$$

If we want to query the intermediate level, we can interpolate. First, bilinear interpolate on the two adjacent levels, and then interpolate again. This is called **Trilinear Interpolation**.

However, using mipmap will cause “overblur” at too far a distance.

Mipmap can only cover axis-aligned rectangular zones. That's why it will cause “overblur”. To solve this problem, we can use **Anisotropic Filtering**.

But it still cannot solve the problem of long images that are mapped diagonally on the texture. There are further methods such as **EWA Filtering**.

3. Applications of Textures

In modern GPUs, texture = memory + range query(filtering).

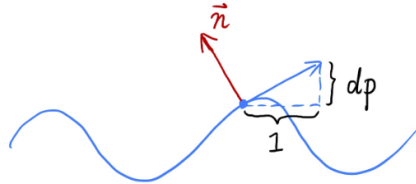
Applications:

- Environment Map
- Environmental Lighting
- Spherical Environment Map
- Cube Map
- Bump / Normal Mapping

- Displacement Mapping
- 3D procedural Noise + Solid Modeling
- Provide Precomputed Shading

Bump / Normal Mapping

- Perturb surface normal per pixel, just to deceive people's eyes.



Perturb the normal in flatland:

- Original surface normal $n(p) = (0, 1)$
- Derivative at p is $dp = c * [h(p + 1) - h(p)]$
- Perturbed normal is $n(p) = (-dp, 1)$ then normalized

Perturb the normal in 3D:

- Original surface normal $n(p) = (0, 0, 1)$
- Derivative at p are
 - $dp/du = c_1 * [h(u + 1) - h(u)]$
 - $dp/dv = c_2 * [h(v + 1) - h(v)]$
- Perturbed normal is $n = (-dp/du, -dp/dv, 1)$ then normalized
- Note that this is in **local coordinates**

Displacement Mapping

It uses the same texture as in bumping mapping, but it actually moves the vertices.

Lecture 10: Geometry 1 (Introduction)

1. Categories of Geometry

- Implicit
 - Algebraic Surface: Combine implicit geometry via Boolean operations.
 - Distance Function: Gradually blend surfaces together using signed distance functions.
 - Level Set: Store a grid of values approximating function, replacing distance functions.
 - Fractal: Exhibit self-similarity, detail at all scales.
 - ...
- Explicit
 - Point Cloud: List of points (x, y, z)
 - Polygon Mesh: Store vertices and polygons (often triangles or quads).
 - Subdivision, NURBS
 - ...

Lecture 11: Geometry 2 (Curves & Surfaces)

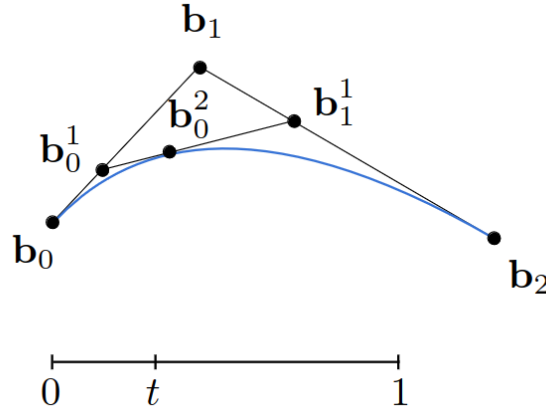
1. Bezier Curves

De Casteljau Algorithm

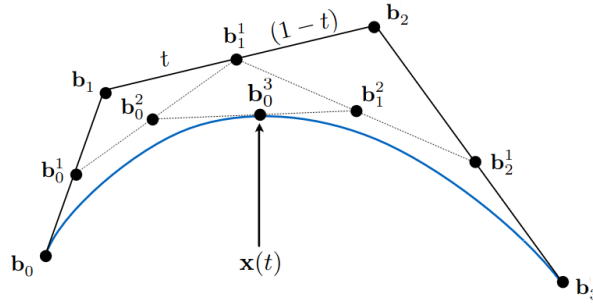
Consider three points, we can generate a unique quadratic bezier curve.

Steps:

1. Lerp(Linear Interpolate) on one line, then repeat on the other.
2. Link the two lerped points.
3. Lerp on the new line, then we get the point on the quadratic bezier curve.
4. Change the value t to generate the whole curve.



When we have four points, to draw a cubic bezier curve, we have a similar method. Each recursion of the algorithm will reduce the degree of the Bezier curve until the final point is found. By changing the value of t , we can obtain Bezier curves of any degree.



Algebraic Formula of Bezier Curves

How we evaluate bezier curves' algebraic formula?

Consider quadratic bezier curve:

$$\begin{cases} b_0^1(t) = (1-t)b_0 + tb_1 \\ b_1^1(t) = (1-t)b_1 + tb_2 \Rightarrow b_0^2(t) = (1-t)^2b_0 + 2t(1-t)b_1 + t^2b_2 \\ b_0^2(t) = (1-t)b_0^1 + tb_1^1 \end{cases}$$

After the promotion we have:

$$b^n(t) = b_0^t = \sum_{j=0}^n b_j B_j^n(t)$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Properties of Bezier Curves

- Interpolates endpoints
 - For cubic bezier curves: $b(0) = b_0, b(1) = b_3$

- Tangent to end segments
 - Cubic case: $b'(0) = 3(b_1 - b_0)$, $b'(1) = 3(b_3 - b_2)$
- Affine transformation property
 - Transform curve by transforming control points
- Convex hull property
 - Curve is within convex hull of control points

Piecewise Bezier Curves

When the degree of the Bezier curve is high, it becomes difficult to adjust and draw. The Bezier curve can be divided into low-degree bezier curves and each curve can be controlled. In practical applications, we usually use cubic Bezier curves.

Continuity:

- C^0 continuity: $a_n = b_0$
- C^1 continuity: $a_n = b_0 = \frac{1}{2}(a_{n-1} + b_1)$
- ...

2. Bezier Surfaces

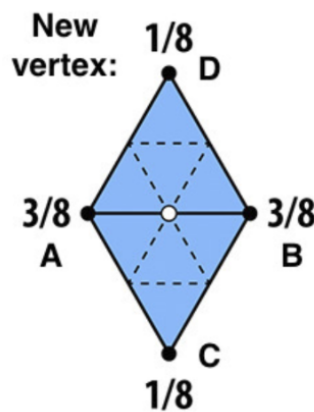
By applying bezier curves on the two axes, we can get a bezier surface.

Lecture 12: Geometry 3 (Mesh Operations & Shadow Mapping)

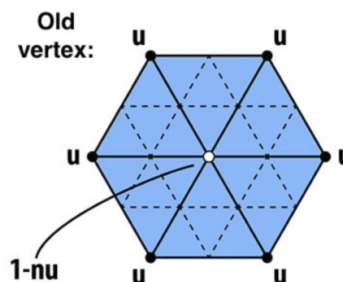
1. Mesh Subdivision

Loop Subdivision

1. Split each triangle into four
2. Assign new vertex positions according to weights



For new vertex like the white dot in the figure above, it is updated to $\frac{3}{8}(A + B) + \frac{1}{8}(C + D)$.

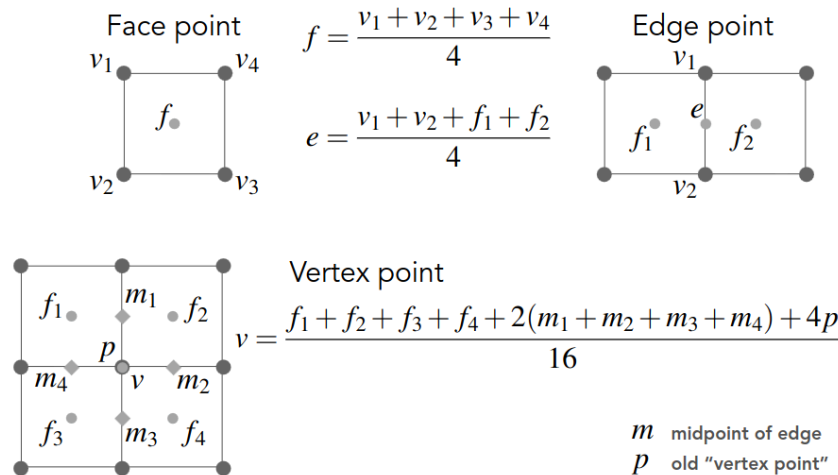


For old vertex like the white dot in the figure above, it is updated to $(1 - n * u) * \text{original_position} + u * \text{neighbor_position_sum}$ (n : vertex degree, u : $\frac{3}{16}$ if $n = 3$, $\frac{3}{8n}$ otherwise).

Catmull-Clark Subdivision (General Mesh)

1. Add vertex in each face
2. Add midpoint on each edge
3. Connect all new vertices

All non-quad faces disappear after just one Catmull-Clark Subdivision.



2. Mesh Simplification

We can use edge collapsing to do mesh simplification.

The problem is how can we select the edge.

Quadratic Error Metrics

Quadratic error: new vertex should minimize its sum of square distance to previously related triangle planes.

When we move one edge, the quadratic error of other edges may change.

We often just use greedy algorithm to select the best edge. In most cases it will have good results.

3. Shadow Mapping

Key idea: the points NOT in shadow must be seen both by the light and by the camera. The light source must be points.

Steps:

1. Render from light: Get the depth image from light source.
2. Render from eye: Project visible points in eye view back to light source.

Problems:

- Hard shadows (point lights only)
- Quality depends on shadow map resolution
- Involves equality comparison of floating point depth values means issues of scale, bias, tolerance.

Lecture 13: Ray Tracing 1 (Whitted-Style Ray Tracing)

1. Basic Ray-Tracing Algorithm

Light Rays

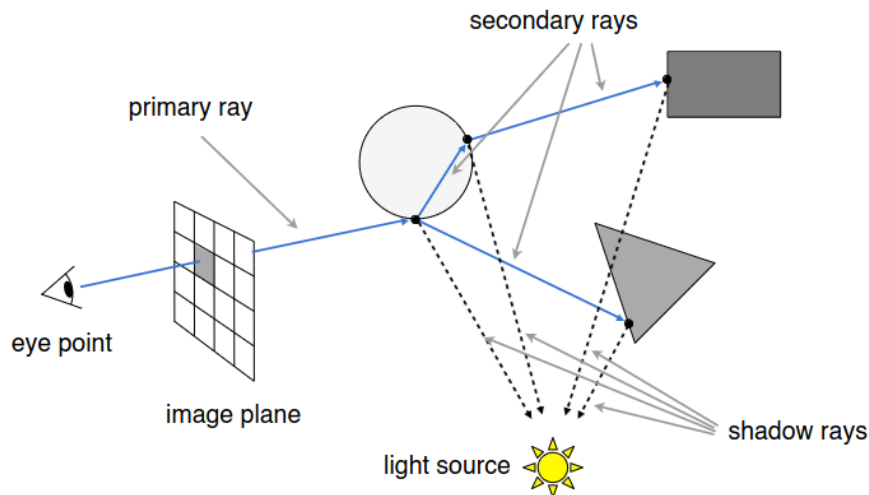
Three ideas about light rays:

1. Light travels in straight lines
2. Light rays do not “collide” with each other if they cross
3. Light rays travel from the light sources to the eye (but here we suppose rays are from the camera)

Ray Tracing is to simulate the light rays.

Recursive (Whitted-Style) Ray Tracing

1. Generate an image by casting one ray per pixel
2. Check for shadows by sending a ray to the light



2. Ray-Surface Intersection

Ray Equation

$$r(t) = o + td, 0 \leq t < \infty$$

Ray Intersection

Sphere:

$$p : (p - c)^2 - R^2 = 0$$

General implicit surface:

$$p : f(p) = 0$$

Substitute ray equation:

$$f(o + td) = 0$$

We solve it for real, positive roots.

How do we intersect a ray with a triangle? Remember Triangle is in a plane.

1. Ray-plane intersection
2. Test if hit point is inside triangle

Plane is defined by normal vector and a point on the plane.

Plane Equation (if p satisfies it, then p is on the plane):

$$p : (p - p') \cdot N = 0$$

Moller Trumbore Algorithm

$$\vec{O} + t\vec{D} = (1 - b_1 - b_2)\vec{P}_0 + b_1\vec{P}_1 + b_2\vec{P}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{S}_1 \cdot \vec{E}_1} \begin{bmatrix} \vec{S}_2 \cdot \vec{E}_2 \\ \vec{S}_1 \cdot \vec{S} \\ \vec{S}_2 \cdot \vec{D} \end{bmatrix}$$

$$\vec{E}_1 = \vec{P}_1 - \vec{P}_0$$

$$\vec{E}_2 = \vec{P}_2 - \vec{P}_0$$

$$\vec{S} = \vec{O} - \vec{P}_0$$

$$\vec{S}_1 = \vec{D} \times \vec{E}_2$$

$$\vec{S}_2 = \vec{S} \times \vec{E}_1$$

Cost = (1 div, 27 mul, 17 add)

3. Accelerating Ray-Surface Intersection

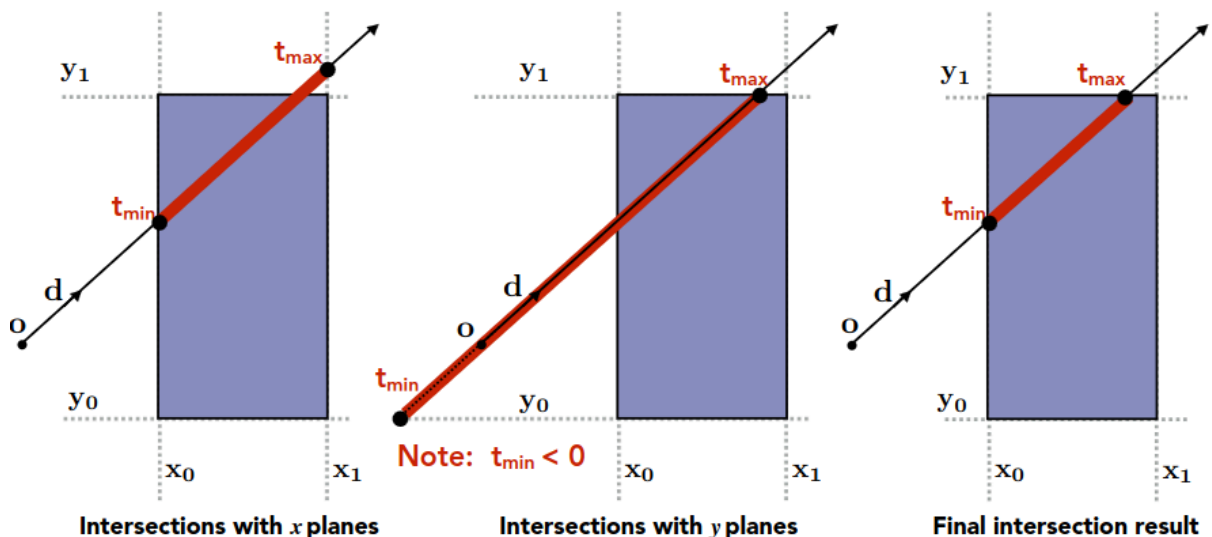
Bounding Volumes

Quick way to avoid intersections: bound complex object with a simple volume

- Object is fully contained in the volume
- If it doesn't hit the volume, it doesn't hit the object
- So test the bounding volume first, then test object if it hits

Box is the intersection of 3 pairs of slabs. We often use an **Axis-Aligned Bounding Box (AABB)**

2D example; 3D is the same! Compute intersections with slabs and take intersection of t_{\min}/t_{\max} intervals



$$t_{\text{enter}} = \max\{t_{\min}\}$$

$$t_{\text{exit}} = \min\{t_{\max}\}$$

If $t_{\text{enter}} < t_{\text{exit}}$, we know the ray stays a while in the box, so they must intersect.

Because ray is not a line, we should check whether t is negative for physical correctness.

- If $t_{\text{exit}} < 0$, the box is “behind” the ray – no intersection.
- If $t_{\text{exit}} \geq 0$ and $t_{\text{enter}} < 0$, the ray’s origin is inside the box – they have intersection.

In summary, ray and AABB intersect if $t_{\text{enter}} < t_{\text{exit}}$ and $t_{\text{exit}} \geq 0$.

Lecture 14: Ray Tracing 2 (Acceleration & Radiometry)

1. Spatial Partitions

We can divide the space into grids to accelerate ray tracing.

1. Find bounding box
2. Create grid
3. Store each object in overlapping cells
4. For each grid cell, test intersection with all objects stored at that cell

Grids work well on large collections of objects that are distributed evenly in size and space.

Better methods:

- Oct-Tree
- KD-Tree
- BSP-Tree
- BVH-Tree

BVH-Tree

Build a BVH-Tree:

1. Find bounding box
2. Recursively split set of objects in two subsets
3. Recompute the bounding box of the subsets
4. Stop when node contains few elements
5. Store objects in each leaf node

Subdivide a node:

1. Choose a dimension to split
2. Always choose the longest axis in node
3. Split node at location of median object

2. Basic Radiometry

Radiometry: Measurement system and units for illumination.

It can perform lighting calculations in a physically correct manner.

Radiant Energy: Radiant energy of electromagnetic radiation.

Radiant Flux: Radiant flux (power) is the energy emitted, reflected, transmitted or received, per unit time.

$$\Phi = \frac{dQ}{dt}$$

Radiant Intensity: The power per unit solid angle emitted by a point light source.

$$I(\omega) = \frac{d\Phi}{d\omega}$$

The unit of I is cd (candela).

Irradiance: The power (perpendicular) per unit area incident on a surface point.

$$E(x) = \frac{d\Phi(x)}{dA}$$

Radiance: The power emitted, reflected, transmitted or received by a surface, per unit solid angle, per projected unit area.

$$L(p, \omega) = \frac{d^2\Phi(p, \omega)}{d\omega dA \cos \theta} = \frac{dE(p)}{d\omega \cos \theta} = \frac{dI(\omega)}{dA \cos \theta}$$

The relation ship between irradiance and radiance:

$$\begin{aligned} dE(p, \omega) &= L_i(p, \omega) \cos \theta d\omega \\ E(p) &= \int_{H^2} L_i(p, \omega) \cos \theta d\omega \end{aligned}$$

Lecture 15: Ray Tracing 3 (Global Illumination)

1. Bidirectional Reflectance Distribution Function (BRDF)

Radiance from direction ω_i turns into the power E that dA receives, then power E will become the radiance to any other direction ω .

$$f_r(\omega_i \rightarrow \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos \theta_i d\omega_i}$$

2. The Rendering Equation

The Reflection Equation:

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega) \cos \theta_i d\omega_i$$

The Rendering Equation:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i$$

Simplified:

$$l(u) = e(u) + \int l(v) K(u, v) dv$$

$K(u, v) dv$ is the kernel of the equation. It refers to **Light Transport Operator**.

Furthermore, we can describe it as:

$$L = E + KL$$

L and E are vectors, K is the light transport matrix.

$$L = E + KL \Rightarrow (I - K)L = E \Rightarrow L = (I - K)^{-1}E$$

(Question: Why can't we calculate $(I - K)^{-1}$ directly?)

Expanding $(I - K)^{-1}$ we can get:

$$L = E + KE + K^2E + K^3E \dots$$

E is directly emitted from light sources.
 KE is directed illumination on surfaces.
 K^2E is one-bounce-indirect illumination.

Lecture 16: Ray Tracing 4 (Monte Carlo Path Tracing)

1. Monte Carlo Integration

We want to solve an integral, but it can be too difficult to solve analytically.

Definite integral: $\int_a^b f(x) dx$

Random variable: $X_i \sim p(x)$

Monte Carlo estimator:

$$\int f(x) dx \approx \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}, X_i \sim p(x)$$

Some notes:

- The more samples, the less variance
- Sample on x , integrate on x

Here is a simple Monte Carlo solution for solving the rendering equation: Assume we uniformly sample the hemisphere, then the **Probability Distribution Function (PDF)** is $\frac{1}{2\pi}$. Now our $f(x)$ is $L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i)$, so we have:

$$L_o(p, \omega_o) \approx \frac{1}{N} \sum_{i=1}^N \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i)}{\text{pdf}(\omega_i)}$$

We can calculate it as follows:

1. Randomly choose N directions within PDF
2. For each ω_i , trace a ray r
 - If r hit the light, add $N^{-1} L_i f_r(n \cdot \omega_i) \text{pdf}(\omega_i)^{-1}$ to L_o
 - If r hit the object at q , add $N^{-1} L_q f_r(n \cdot \omega_i) \text{pdf}(\omega_i)^{-1}$ to L_o . L_q is calculated in the same way at this step, which is a recursive process.
3. L_o is the result

2. Path Tracing

The algorithm above can't wholly solve the problem.

- The number of rays will explode when the bounce count goes up. The process can be too complex.
- The recursive algorithm will never stop.

For the first problem, we can assume that only 1 ray is traced at each shading point. We call this **Path Tracing**. This will cause noise, but we can just trace more paths through each pixel and average their radiance.

For the second, a naive method is to limit the bouncing count, but this will cut energy, thus affects the rendering effect.

The Solution is called **Russian Roulette (RR)**.

Previously, we always shoot a ray at a shading point and get the shading result L_o .

Suppose we have set a probability $P \in (0, 1)$.

- With probability P , shoot a ray and return the shading result divided by P : L_o/P

- With probability $1 - P$, don't shoot a ray and you'll get 0

Finally, we can get L_o :

$$E = P * \frac{L_o}{P} + (1 - P) * 0 = L_o$$

Now we already have a correct version of path tracing, but it's not really efficient.

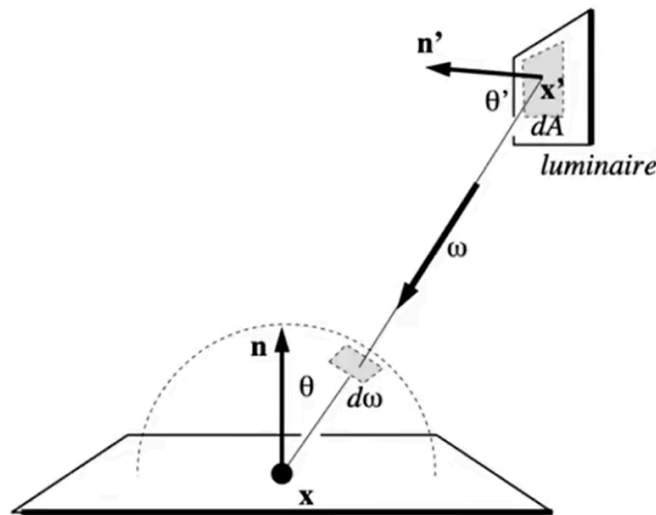
When we uniformly sample the hemisphere at the shading point, a lot of rays are “wasted”. Given the situation, we can sample the light, therefore no rays are “wasted”.

Assume uniformly sample on the light, pdf = $\frac{1}{A}$. We are sampling on the light, but the rendering equation integrates on the solid angle, so we should rewrite the rendering equation.

$$d\omega = \frac{dA \cos \theta'}{\|x' - x\|^2}$$

The rewritten rendering equation:

$$L_o(x, \omega_o) = \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos \theta \cos \theta'}{\|x' - x\|^2} dA$$



Now $f(x)$ is everything inside, pdf is $\frac{1}{A}$.

Previously, we assume the light is “accidentally” shot by uniform hemisphere sampling. Now, we consider the radiance coming from two parts:

1. light source (direct, no need to have RR)
2. other reflectors (indirect, RR)

Lecture 17: Materials and Appearances

1. Diffuse / Lambertian Material

Suppose the incident lighting is uniform:

$$L_o(\omega_o) = \int_{H^2} f_r L_i(\omega_i) \cos \theta_i d\omega_i = f_r L_i \int_{H^2} \cos \theta_i d\omega_i = \pi f_r L_i$$

We can define albedo(color) ρ , then $f_r = \frac{\rho}{\pi}$.

2. Glossy Material

3. Ideal Reflective / Refractive Material

Some properties to consider:

- Snell's Window / Circle
- Fresnel Reflection / Term

Fresnel Term:

$$R_s = \left| \frac{n_1 \cos \theta_i - n_2 \cos \theta_t}{n_1 \cos \theta_i + n_2 \cos \theta_t} \right|^2 = \left| \frac{n_1 \cos \theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2}}{n_1 \cos \theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2}} \right|^2,$$

$$R_p = \left| \frac{n_1 \cos \theta_t - n_2 \cos \theta_i}{n_1 \cos \theta_t + n_2 \cos \theta_i} \right|^2 = \left| \frac{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2} - n_2 \cos \theta_i}{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2} + n_2 \cos \theta_i} \right|^2.$$

$$R_{\text{eff}} = \frac{1}{2} (R_s + R_p).$$

It seems too complex, and we have **Schlick's Approximation**:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

4. Microfacet Material

Microfacet Theory

- Macroscale: flat & rough
- Microscale: bumpy & specular

The distribution of microfacets' normal will affect its BRDF:

- Concentrated \Leftrightarrow Glossy
- Spread \Leftrightarrow Diffuse

5. Isotropic / Anisotropic Materials

For anisotropic BRDFs, reflection depends on azimuthal angle ϕ .

6. Properties of BDRFs

- Non-negativity
- Linearity
- Reciprocity Principle
- Energy Conservation

Lecture 18: Advanced Topics in Rendering

1. Unbiased Light Transport Methods

The expected value of an unbiased estimator will always be the correct value, no matter how many samples are used.

Bidirectional Path Tracing (BDPT)

- Trace sub-paths from both the camera and the light

- Connect the end points from both sub-paths

Advantage:

- Suitable if the light transport is complex on the light's side

Disadvantage:

- Difficult to implement and quite slow

Metropolis Light Transport (MLT)

- A Markov Chain Monte Carlo (MCMC) application
- Jumping from the current sample to the next with some PDF

Advantage:

- Very good at locally exploring difficult light paths

Disadvantage:

- Difficult to estimate the convergence rate
- Does not guarantee equal convergence rate per pixel, so usually produces “dirty” results. Therefore, usually not used to render animations.

2. Biased Light Transport Methods

Photon Mapping

- Stage 1: Photon Tracing
 - Emitting photons from the light source, bouncing them around, then recording photons on diffuse surfaces
- Stage 2: Photon Collection
 - Shoot sub-paths from the camera, bouncing them around, until they hit diffuse surfaces
- Stage 3: Calculation – Local Density Estimation
 - Idea: Areas with more photons should be brighter
 - For each shading point, find the nearest N photons. Take the surface area they cover

Advantage:

- Very good at handling Specular-Diffuse-Specular (SDS) paths and generating caustics

Disadvantage:

- Biased (but consistent)

Vertex Connection and Merging

- A combination of BDPT and Photon Mapping
- Key idea
 - Let's not waste the sub-paths in BDPT if their end points cannot be connected but can be merged
 - Use photon mapping to handle the merging of nearby “photons”

Instant Radiosity (IR)

- Key idea: Lit surfaces can be treated as light sources
- Approach:
 - Shoot light sub-paths and assume the end point of each sub-path is a Virtual Point Light (VPL)
 - Render the scene as usual using these VPLs

Advantage:

- Fast and usually gives good results on diffuse scenes

Disadvantage:

- Spikes will emerge when VPLs are close to shading points

- Cannot handle glossy materials

3. Advanced Appearance Modeling

Non-Surface Models

- Participating Media
 - Example:
 - Cloud
 - Fog
 - Rendering:
 - Randomly choose a direction to bounce
 - Randomly choose a distance to go straight
 - At each “shading point”, connect to the light
- Hair/Fur/Fiber
 - Model:
 - Kajiya-Kay Model
 - Marschner Model
 - Double Cylinder Model
- Granular Material

Surface Models

- Translucent Material
 - Example:
 - Jade
 - Jellyfish
 - Subsurface Scattering: We use BSSRDF, which can be seen as a generalization of BRDF.
- Cloth

Procedural Appearance

Lecture 19: Cameras, Lenses and Light Fields

1. Some Basic Concepts

Pinhole Camera Formation

This will get a sharp image.

Field of View (FOV)

$$\text{FOV} = 2 \arctan\left(\frac{h}{2f}\right)$$

Exposure

Exposure = time \times irradiance

$$H = T \times E$$

Exposure Control Factors in Photography:

- Aperture Size: Change the f-stop by opening/closing the aperture
- Shutter Speed: Change the duration the sensor pixels integrate light
- ISO: Change the amplification (analog and/or digital) between sensor values and digital image values

Because it takes time for the shutter to open/close, **motion blur** will occur if the scene is moving. Different parts of photo are taken at different times, this will cause **rolling shutter effect**.

Thin Lens Approximation

Ideal Thin Lens:

1. All parallel rays entering a lens pass through its focal point.
2. All rays through a focal point will be in parallel after passing the lens.
3. Focal length can be arbitrarily changed.

Defocus Blur

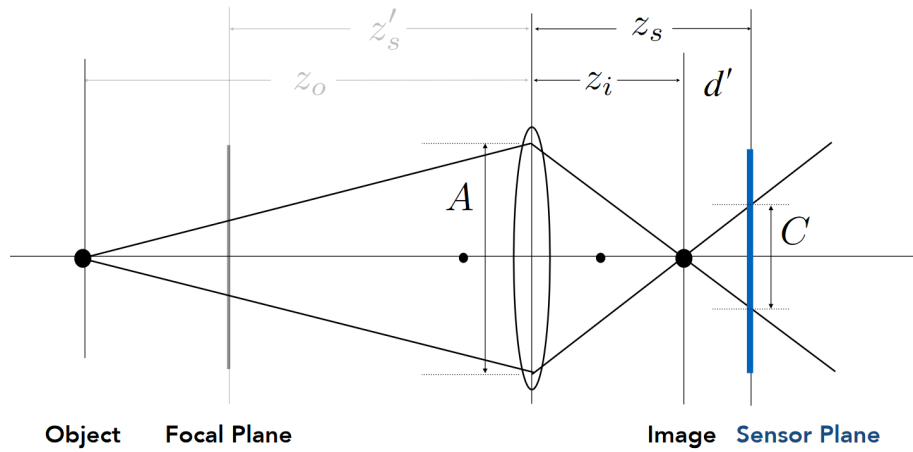
Circle of confusion is proportional to the size of the aperture.

$$\frac{C}{A} = \frac{d'}{z_i} = \frac{|z_s - z_i|}{z_i}$$

$$\Rightarrow C = A \frac{|z_s - z_i|}{z_i} = \frac{f}{N} \frac{|z_s - z_i|}{z_i}$$

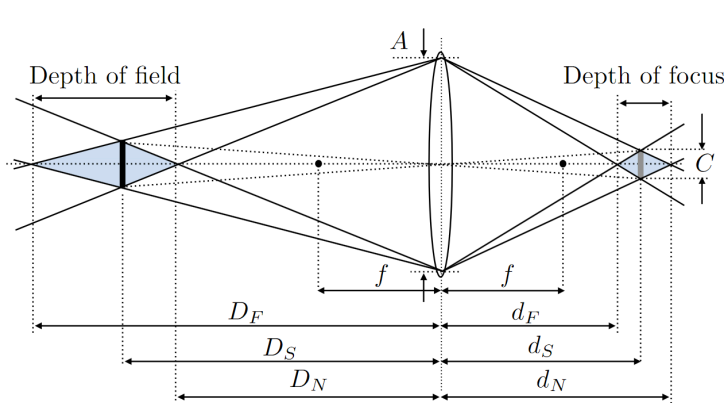
So if we want to take a sharper photo, we may use a smaller aperture.

F-Number = Focal Length / Aperture Diameter



Depth of Field

Depth of Field is the depth range in a scene where the corresponding CoC is considered small enough.



$$\frac{d_N - d_S}{d_N} = \frac{C}{A}$$

$$\frac{d_S - d_F}{d_F} = \frac{C}{A}$$

$$N = \frac{f}{A}$$

$$\frac{1}{D_F} + \frac{1}{d_F} = \frac{1}{f}$$

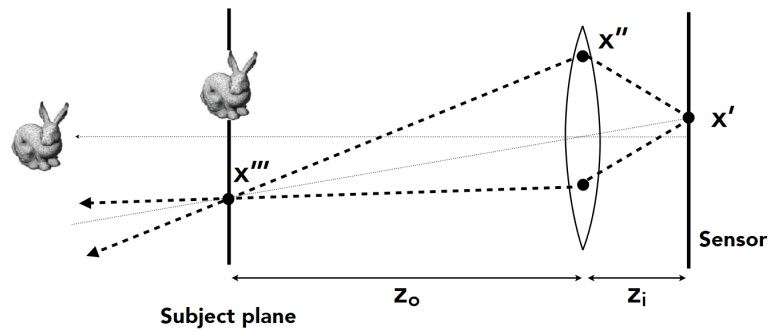
$$\frac{1}{D_S} + \frac{1}{d_S} = \frac{1}{f}$$

$$\frac{1}{D_N} + \frac{1}{d_N} = \frac{1}{f}$$

$$\text{DOF} = D_F - D_N$$

$$D_F = \frac{D_S f^2}{f^2 - NC(D_S - f)} \quad D_N = \frac{D_S f^2}{f^2 + NC(D_S - f)}$$

2. Ray Tracing for Defocus Blur (Thin Lens Model)



Rendering:

- For each pixel x' on the sensor:
 - Sample random points x'' on lens plane
 - The ray passing through the lens will hit x''' (using the thin lens formula)
 - Estimate radiance on ray $x'' \rightarrow x'''$

3. Light Field / Lumigraph

The set of all things that we can see can be described by the **Plenoptic Function**:

$$P(\theta, \varphi, \lambda, t, V_x, V_y, V_z)$$

Two planes can describe a light field (Two points can determine a line). One plane holds s, t , and the other holds u, v .

Based on this principle, people created light field cameras, which supports post-adjustment of parameters.

Lecture 20: Color and Perception

1. What is Color?

Physical Basis of Color

White light is composed of light of different frequencies (wavelengths).

Spectral Power Distribution describes the power distribution of light.

Biological Basis of Color

Human eyes are like cameras. They have two types of photoreceptor cells: rods and cones. Rods perceive only shades of gray, but no color. Cones have three types: S, M and L. They can provide the sensation of color.

Color is a phenomenon of human perception. It is not a universal property of light.

Metamerism

Two light sources of different spectral may produce the same color (Color in humans' eyes is produced by the integral of SPD and eye response function). This is the theory behind color matching.

2. Color Reproduction / Matching

Given a set of primary lights, each with its own spectral distribution (e.g. R,G,B display pixels):

$$S_R(\lambda), S_G(\lambda), S_B(\lambda)$$

Adjust the brightness of these lights and add them together:

$$RS_R(\lambda) + GS_G(\lambda) + BS_B(\lambda)$$

In an **Additive Color System**, maybe some colors cannot be reproduced by mix the primary lights.

Color Spaces

Gamut is the set of chromaticities generated by a set of color primaries. Different color spaces represent different ranges of colors.

A universal Color Space is **CIE XYZ**. It is an artificially defined matching function.

Usually we use **Standardized RGB (sRGB)** color space.

Another widely used color space is **HSV (Hue-Saturation-Value)**.

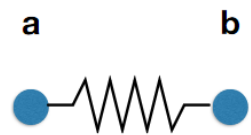
CIE LAB is a commonly used color space that strives for perceptual uniformity.

In real life, for example, when printing, the subtractive color system is used. A color space called **CMYK** is widely used. Among them, K stands for key(black). C(Cyan), M(Magenta), Y(Yellow) can generate K. Due to cost reasons, K is still used.

Lecture 21: Animation

1. MASS Spring System

A Simple Spring:



$$f_{a \rightarrow b} = k_s(b - a)$$

$$f_{b \rightarrow a} = -f_{a \rightarrow b}$$

- Force pulls points together
- Strength proportional to displacement (Hooke's Law)
- k_s is a spring coefficient: stiffness

Problem: This Spring wants to have zero length.

Non-Zero Length Spring:

$$f_{a \rightarrow b} = k_s \frac{b - a}{\|b - a\|} (\|b - a\| - l)$$

Problem: This Spring will oscillate forever.

Internal Damping for Spring:

Note: In simulation, we use dot notation for derivatives.

$$f_b = -k_d \frac{b - a}{\|b - a\|} (\dot{b} - \dot{a}) \cdot \frac{b - a}{\|b - a\|}$$

Relative velocity of b, assuming a is static (vector)
 ↓
 Damping force applied on b
 ↑
 Relative velocity projected to the direction from a to b (scalar)
 Direction from a to b

Particle System

Model dynamical systems as collections of large numbers of particles.

Each particle's motion is defined by a set of physical (or non-physical) forces.

Challenges:

- May need many particles (e.g. fluid)
- May need acceleration structures (e.g. to find nearest particles for interactions)

Algorithm:

- Create new particles if needed
- Calculate forces on each particle
- Update each particle's position and velocity
- Remove dead particles if needed
- Render particles

2. Kinematics

Forward Kinematics

Strengths:

- Direct control is convenient
- Implementation is straightforward

Weaknesses:

- Animation may be inconsistent with physics
- Time consuming for artists

Inverse Kinematics

Animator provides position of end-effector, and computer determines joint angles that satisfy constraints. Computing it is hard, so usually we apply gradient descent (or Newton's method, or other optimization procedures).

3. Rigging

Rigging is a set of higher level controls on a character that allow more rapid & intuitive modification of poses, deformations, expressions, etc.

4. Motion Capture

- Record real-world performances
- Extract pose as a function of time from the data collected

Strengths

- Can capture large amounts of real data quickly
- Realism can be high

Weaknesses

- Complex and costly set-ups
- Captured animation may not meet artistic needs, requiring alterations

Lecture 22: Animation Cont.

1. Single Particle Simulation

First study motion of a single particle. Later, generalize to a multitude of particles.

To start, assume motion of particle determined by a velocity vector field that is a function of position and time: $v(x, t)$

$$\frac{dx}{dt} = \dot{x} = v(x, t)$$

To compute position, we can use **Euler's Method (a.k.a Forward Euler, Explicit Euler)**:

$$x_{t+\Delta t} = x_t + \Delta t \dot{x}_t$$

$$\dot{x}_{t+\Delta t} = \dot{x}_t + \Delta t \ddot{x}_t$$

- Simple iterative method
- Commonly used
- Very inaccurate for large time steps
- Most often goes unstable

Some Methods to Combat Instability:

- Midpoint Method
 1. Compute Euler step
 2. Compute derivative at midpoint of Euler step
 3. Update position using midpoint derivative
- Adaptive Step Size
 - Repeat until error is below threshold:
 1. Compute x_T an Euler step, size T
 2. Compute $x_{T/2}$ two Euler steps
 3. If error is larger than threshold, repeat with smaller step size
- Implicit Euler Method
 - Use derivatives in the future, for the current step
 - $x_{t+\Delta t} = x_t + \Delta t \dot{x}_{t+\Delta t}$
 - $\dot{x}_{t+\Delta t} = \dot{x}_t + \Delta t \ddot{x}_{t+\Delta t}$
 - Hard to compute
 - Use root-find algorithm, e.g. Newton's method
 - Offer much better stability
- Position-Based / Verlet Integration
 - Fast and simple
 - Not physically based, dissipates energy

Runge-Kutta Families: A family of advanced methods for solving **Ordinary Differential Equation (ODEs)**

- Especially good at dealing with non-linearity
- It's order-four version is the most widely used, a.k.a. RK4

(TODO: Make it clear)

3. Rigid Body Simulation

- Similar to simulating a particle
- Just consider a bit more properties

$$\frac{d}{dt} \begin{pmatrix} X \\ \theta \\ \dot{X} \\ \omega \end{pmatrix} = \begin{pmatrix} \dot{X} \\ \omega \\ F/M \\ \Gamma/I \end{pmatrix}$$

X : positions

θ : rotation angle

ω : angular velocity

F : forces

Γ : torque

I : momentum of inertia

4. Fluid Simulation

Key idea:

- Assume water is composed of small rigid-body spheres
- Assume water cannot be compressed

Two different views to simulating large collections of matters: Lagrangian and Eulerian