# Shading

2021年7月12日 18:42

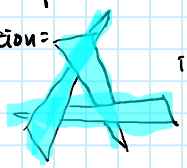
## § Shading 1 (Illumination, Shading and Graphics Pipeline)

Content:
- Visibility / occlusion
  - z-buffering
- Shading
  - Illumination & Shading
  - Graphics Pipeline.

### 1. Painter's algorithm

Paint from back to front. overwrite in the frame buffer

Exception:



impossible to paint with Painter's algorithm

### 2. Z-buffer

Idea:
- Store current min. z-value for each sample (pixel)
- Needs an additional buffer for depth values
  - frame buffer stores color values
  - depth buffer (z-buffer) stores depth.

IMPORTANT: suppose z- is always positive
smaller z → closer
larger z → further)

### 3. Z-buffer algorithm

During rasterization:

```
for each triangle T:
    for each sample (x,y,z) in T:
        if (z < zbuffer[x,y])
            framebuffer[x,y] = rgb
            Zbuffer[x,y] = z        // update z-buffer
        else
            )
```

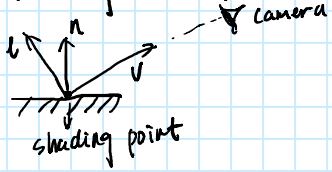
### 4. Complexity

O(n) for n triangles (assume const coverage)

### 5. (Blinn-Phong Reflectance Model) Shading Model

Specular highlights } 高光
Diffuse reflection } 漫反射
Ambient lighting } 环境光

### 6. Shading is local



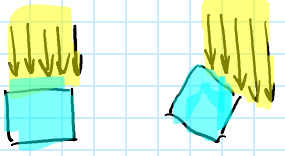

- View direction v
- surface normal n
- light direction l
- Surface parameters (color, shininess ...)

Notice shadows ≠ shading
着色只考虑该点，不考虑其他物体遮盖 (local)

Notice shadows ≠ shading
着色只考虑该点，不考虑其他物体遮盖 (local)

1. Diffuse reflection.

Lambert's cosine law



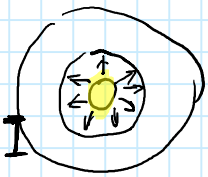

光强 → 能量.

light per unit area is proportional to $\cos\theta = \vec{l}\cdot\vec{n}$

Light Falloff

$4\pi r^2 \cdot E(r) = \text{const}$

# define intensity at r=1 is I, then $\text{Intensity}(r) = I/r^2$



Lambertian Shading → independent of view direction $\vec{v}$

$L_d = k_d (I/r^2) \max(0, \vec{n}\cdot\vec{l})$ → cosine law

$L_d$: diffusely reflected light; $k_d$: diffuse reflected light

$k_d$ decides the overall brightness

§ Shading 2 (Shading. Pipeline and Texture Mapping)

1. Specular Term

v close to mirror direction ⇔ half vector near normal


$h = \text{bisector}(\vec{v}, \vec{l}) = \frac{\vec{v}+\vec{l}}{\|\vec{v}+\vec{l}\|}$


半程向量.

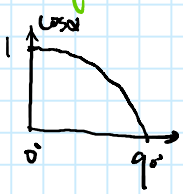
$L_s = k_s (I/r^2) \max(0, \cos\alpha)^p$

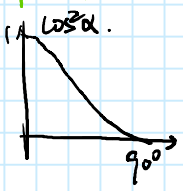
$= k_s (I/r^2) \max(0, \vec{n}\cdot\vec{h})^p$

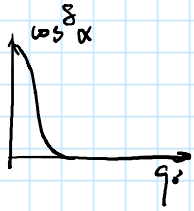
specularly reflected light; $k_s$ coefficient

normally $p = 100 - 200$

$p \gg 1$, size of bright area.

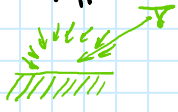



2. Ambient Term

Approximation $L_a = k_a I_a$



reflected ambient light

ensure there's not a complete black plane.

3. $L = L_a + L_d + L_s$

4. Shading Frequencies

Flat shading (for 同着色)

4. Shading Frequencies

① Flat shading 面着色

- Triangle face is flat — one normal vector
- Not good for smooth surfaces

② Gouraud shading 顶点着色

- Interpolated colors from vertices across triangles
- Each vertex has a normal vector. how?

③ Phong shading 逐pixel

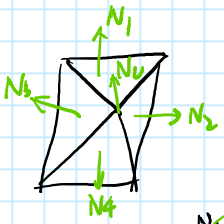
- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel.

When Num Vertices ↑ Flat shading has similar effect as Phong shading

5. Define Per-Vertex Normal Vectors

average surrounding face normals

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

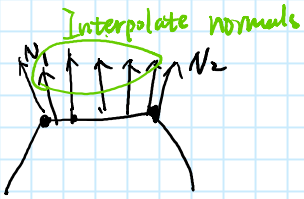



Modification: add weights to each normal (area of each face)

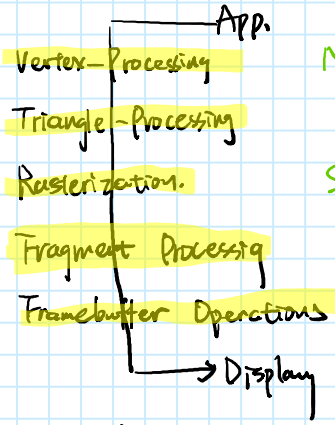
Barycentric interpolation of vertex normals

↓

Introduced later




6. Graphics Pipeline (Real-time Rendering Pipeline)




7. Shader Programs

- Program vertex and fragment processing stages
- Describe operation on a single vertex or fragment.

↓ Thus, no need for loops

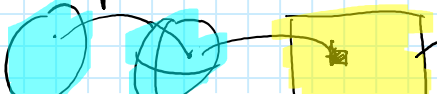
GLSL = shader Language.

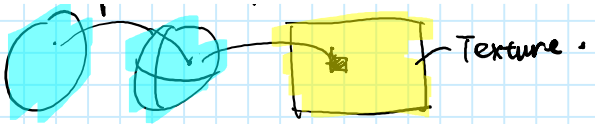
8. Shader Toy Program = Focus on shading

§ Texture Mapping

1. Surface of 3D objects are 2D

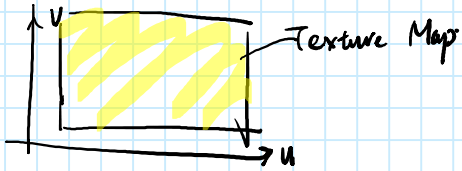
Map 3D surface to 2D






Map 3D to 2D

2. Texture Coordinates (u,v)

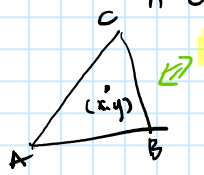



For convenient $(u,v) \in [0,1]^2$

## § Shading 3 (Texture Mapping Cont.)

1. Vertices shading = Interpolate is key
2. Barycentric Coordinates

A coordinate for triangles $(\alpha, \beta, \gamma)$

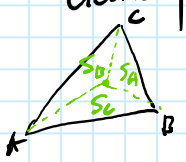



$(x,y) = \alpha A + \beta B + \gamma C$

$(\alpha + \beta + \gamma = 1)$

Inside the triangle $\alpha, \beta, \gamma$ are non-negative.

Geometry Perspective - area propotion.




$$\alpha = \frac{S_A}{S_A + S_B + S_C}$$

$$\beta = \frac{S_B}{S_A + S_B + S_C}$$

$$\gamma = \frac{S_C}{S_A + S_B + S_C}$$

Formulas

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

$\beta =$ ……

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

3. Interpolate values at vertices

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

eg. colors

Notice: Barycentric coordinates may change under projection.

Interpolation is applied before projection. in 3D space

4. Simple Texture Mapping: Diffuse Color

For each screen sample (x,y):

(u,v) = evaluate texture coordinates at (x,y)

texcolor = texture.sample(u,v)

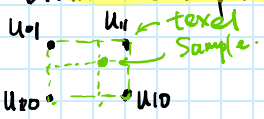
Set sample's color to texcolor

↑ $k_d$ in Bling Phong Model

5. Q1: What if texture is too small ⇒ Texture Magnification.

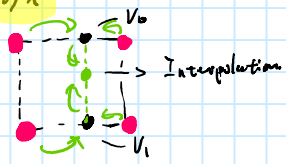
#define texel = pixel on texture

## Bilinear Interpolation.




Linear Interpolation (1D)

$lerp(x, v_0, v_1) = v_0 + (v_1 - v_0)x$




Two helper lerps

$u_0 = lerp(s, u_{00}, u_{10})$

$u_1 = lerp(s, u_{01}, u_{11})$

Final vertical lerp

$f(x,y) = lerp(t, u_0, u_1)$

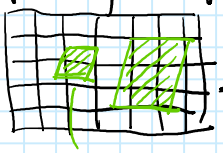
Bilinear Interpolation = poor performance on thin edges

⇓

Bicubic Interpolation. = use near 16 texels to interpolate

b. $Q_2$ = What if texture is too large? ⇒ Even more severe problem

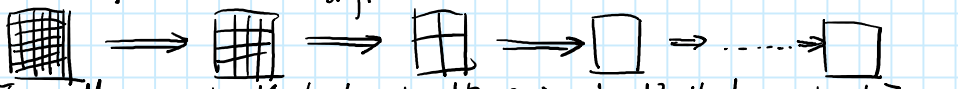
Point Query vs. Range Query ⇕ equivalent to Aliasing




screen. small screens can't cover enough information.

7. Mipmap

Allowing fast, approx, square only! range queries




A total of $\log_2 N$ layers

总计算量为原图的 $\frac{4}{3}$

Computing Mipmap Level D


Suppose square on Screen ⇒ a square on Texture ⇕ Actually not!

假设 Texture 上的映射也为正方形

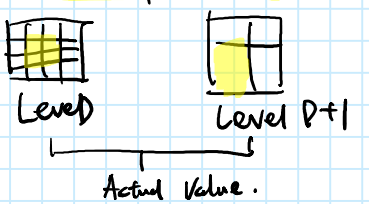
$D = \log_2 L$, $L = \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{du}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$

求出在哪一个 Level, LXL 为一个 texel

然后直接查 Level D 的 Mipmap 对应像素即可

What if D is not an integer?

⇒ Interpolation!




} Trilinear Interpolation.

8. Mipmap limitations:

Detail loss when distance is too far Over blur.

Why? : square. approx.

Solution: ① Anisotropic Filtering 各项异性过滤 Ripmap

Solution: ① Anisotropic Filtering 各项异性过滤 Ripmap

Overall 3 times Computation Amount.

← Vertically and Horizontally Compressed Pictures

Expand from square to rectangles

Still not good Enough if rectangles are diagonal.

② EWA Filtering

6. Application of textures

In GPUs, texture = memory + range query

Application → Environment Map

texture reflection Utah teapot.

环境光认为在无穷远处，只需记录方向信息即可

→ Environment Lighting

Spherical Map

将环境光信息放在球上，然后展开成平面信息

Cube Map

Sphere

Map a sphere to six planes

↓

6 texture map

→ 凹凸贴图

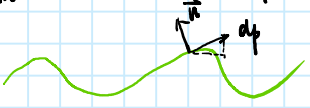
texture can store height information.

Bump / normal mapping

Suppose origined surface normal n(p) = (0,1)

Derviate at p is dp = c * [h(p+1) - h(p)]

normal is then n(p) = (-dp, 1), normalized(.)



In 3D at

$n(p) = (0,0,1)$

$n = (-dp/du, -dp/dv, 1)$

$\frac{dp}{du} = c1 * [h(u+1) - h(u)]$

$\frac{dp}{dv} = c2 * [h(v+1) - h(v)]$

A more modern approach - Displacement Mapping 在边界上会有更好的效果

Actually make changes to geometry, move the vertices

Dynamic tessellation.

Displacement mapping requires large amount of triangles in the model.

DirectX use dynamic methods to test the right amount of triangles

Plus, reducing computation

displacement mapping requires [illegible] triangles for the model.
DirectX use dynamic methods to test the right amount of triangles
-thus, reducing computation

→ 3D Procedural Noise

→ provide precomputed shading