计算机图形学入门分享

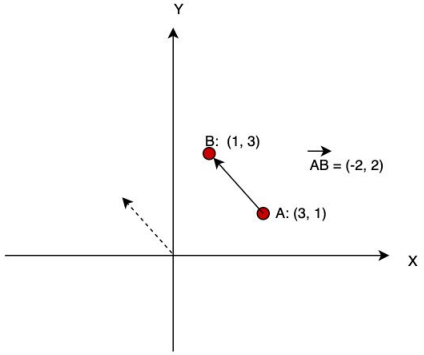
Tony

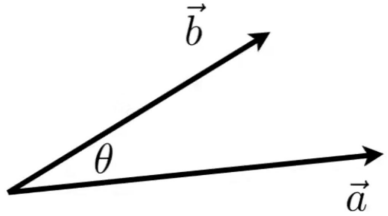
基础知识：数学 & 物理

1. 线性代数（Linear algebra）
2. 微积分（Calculus）
3. 统计（Statistics）
4. 光学（Optics）
5. 力学（Mechanics）
6. 信号处理（Signal processing）

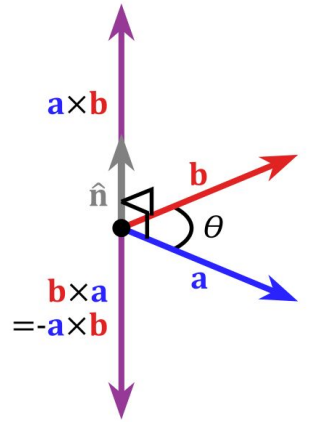
向量（Vectors）

- 点（Point）
- 方向（Direction）
- 点乘（Dot/Scalar Product）
- 叉乘（Cross Product）





$$\vec{a} \cdot \vec{b} = \|\vec{a}\|\|\vec{b}\| \cos\theta$$



矩阵（Matrix）

- 相乘（Mulitiplication）
- 转置（Transpose）
- 单位矩阵（Identity）
- 逆（Inverse）

$$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix} \times \begin{bmatrix}10&11\\20&21\\30&31\end{bmatrix} = \begin{bmatrix}1x10+2x20+3x30 & 1x11+2x21+3x31\\ 4x10+5x20+6x30 & 4x11+5x21+6x31\end{bmatrix} = \begin{bmatrix}10+40+90 & 11+42+93\\ 40+100+180 & 44+105+186\end{bmatrix} = \begin{bmatrix}140&146\\320&335\end{bmatrix}$$

Transpose

$$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \quad \begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$$

$$\begin{bmatrix}1&0\\0&1\end{bmatrix}$$

2 x 2

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

3 x 3

Inverse of a Matrix

If $A = \begin{pmatrix}a&b\\c&d\end{pmatrix}$ $A' = \frac{1}{ad-bc}\begin{pmatrix}d&-b\\-c&a\end{pmatrix}$ $AA' = \begin{pmatrix}1&0\\0&1\end{pmatrix}$

inverse of A

determinant

Identity matrix

向量运算的矩阵表示

- Dot product?

$$\vec{a} \cdot \vec{b} = \vec{a}^T \vec{b}$$

$$= \begin{pmatrix} x_a & y_a & z_a \end{pmatrix} \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = \begin{pmatrix} x_a x_b + y_a y_b + z_a z_b \end{pmatrix}$$

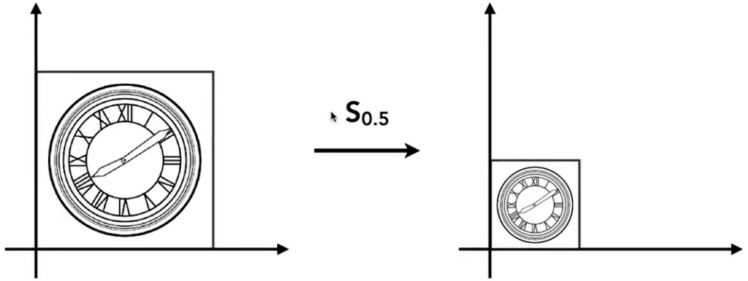
- Cross product?

$$\vec{a} \times \vec{b} = A^* b = \begin{pmatrix} 0 & -z_a & y_a \\ z_a & 0 & -x_a \\ -y_a & x_a & 0 \end{pmatrix} \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix}$$

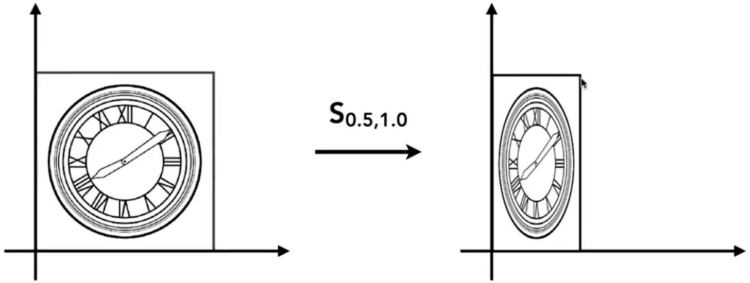
变换（Transformation）

- 缩放（Scale）
- 反射（Reflect）
- 裁剪（Shear）
- 旋转（Rotate）
- 平移（Translate）

缩放（Scale）

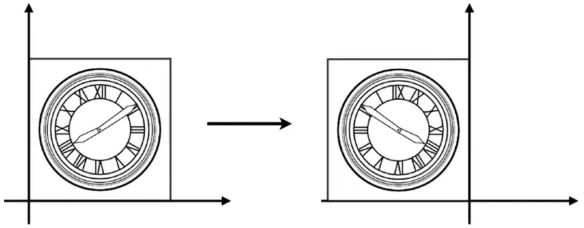


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

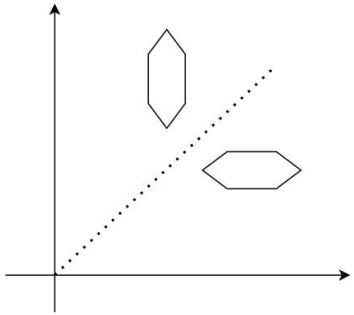


?

反射（Reflect）

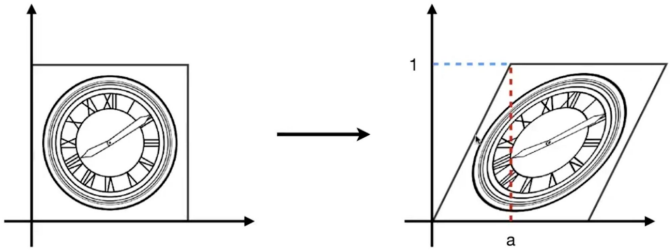


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

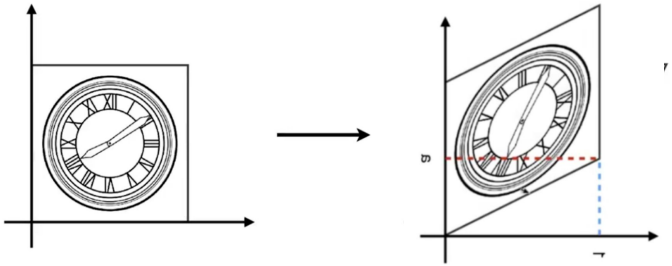


?

剪切（Shear）

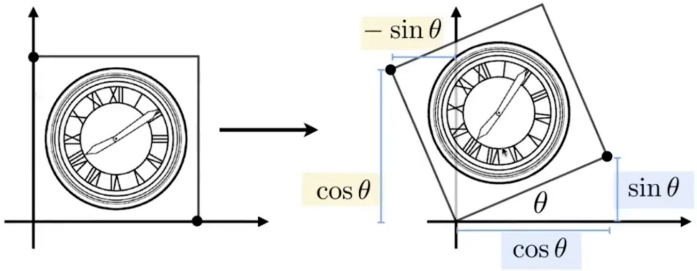


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



?

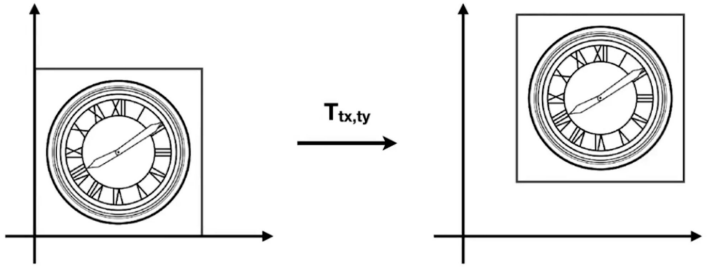
旋转（Rotate）



$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

以点（x0, y0）为中心旋转呢？

平移（Translate）



$$x' = x + t_x$$
$$y' = y + t_y$$

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

仿射变换 & 齐次坐标（Affine Transformations & Homogenous Coordinates）

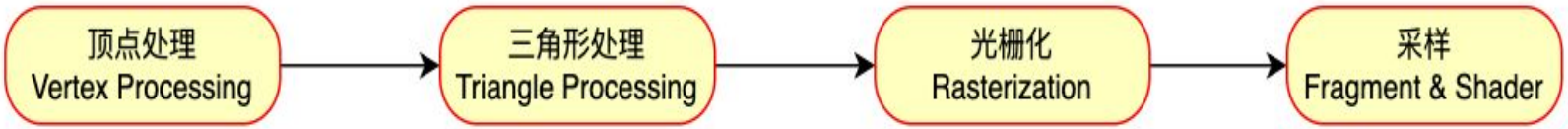
Affine map = linear map + translation

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

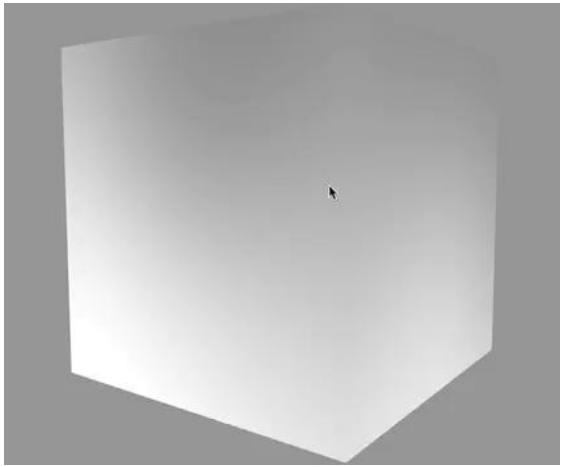
Using homogenous coordinates:

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

渲染管道（Graphics Pipeline(Real-time Rendering)）



3D 文件 obj：顶点（Vertices in 3D Space）



```
 1 v -1 -1  1
 2 v  1 -1  1
 3 v  1  0  1
 4 v  0  1  1
 5 v -1  1  1
 6 v  1  1  0
 7 v -1 -1 -1
 8 v  1 -1 -1
 9 v  1  1 -1
10 v -1  1 -1
11
12 f 1/-1/-1 2/-1/-1 3/-1/-1 4/-1/-1 5/-1/-1
13 f 4/-1/-1 3/-1/-1 6/-1/-1
14 f 1/-1/-1 7/-1/-1 8/-1/-1 2/-1/-1
15 f 1/-1/-1 5/-1/-1 10/-1/-1 7/-1/-1
16 f 5/-1/-1 4/-1/-1 6/-1/-1 9/-1/-1 10/-1/-1
17 f 2/-1/-1 8/-1/-1 9/-1/-1 6/-1/-1 3/-1/-1
18 f 7/-1/-1 10/-1/-1 9/-1/-1 8/-1/-1
19
```

3D 模型：african_head.obj

v -0.000581696 -0.734665 -0.623267 Vertex

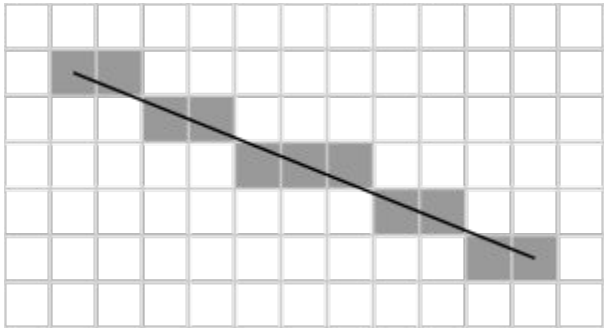
vt 0.350 0.002 0.000 Texture

vn -0.001 0.661 0.751 Normal

f 32/9/32 33/10/33 34/11/34 Face

```
class Model:

    _vert_flag = "v"
    _face_flag = "f"

    def __init__(self, filename: str) -> None:

        self._verts = []
        self._faces = []

        with open(filename, "r") as f:
            for l in f:
                segs = re.split(r"[ ]+", l)
                if len(segs) != 4:
                    continue
                if segs[0] == Model._vert_flag:
                    self._verts.append(tuple([float(i) for i in segs[1:]]))
                elif segs[0] == Model._face_flag:
                    self._faces.append(
                        tuple(int(seg.split("/")[0]) - 1 for seg in segs[1:])
                    )
```

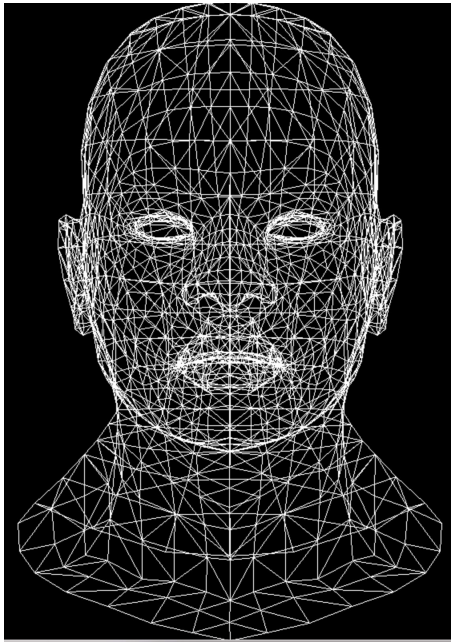
画三角形（Triangle）



Bresenham's line algorithm

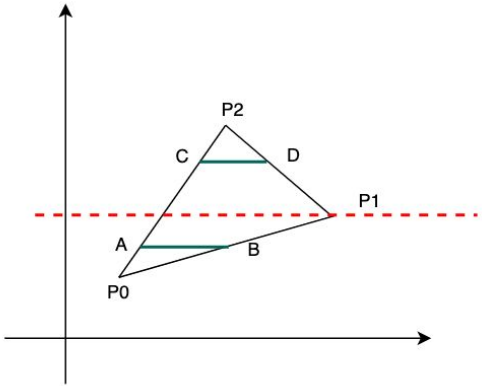
```python
def line_4(x0, y0, x1, y1: int, image: CoordinateImage, color):
    step = abs(x1 - x0) > abs(y1 - y0)
    if not step:
        x0, y0 = y0, x0
        x1, y1 = y1, x1
    if x0 > x1:
        x0, x1 = x1, x0
        y0, y1 = y1, y0

    dx = x1 - x0
    dy = abs(y1 - y0)
    de = dy / dx
    err = 0.0
    y = y0
    y_direction = 1 if y1 > y0 else -1
    for x in range(x0, x1 + 1):
        if step:
            image.set(x, y, color)
        else:
            image.set(y, x, color)
        err += de
        if err > 0.5:
            y += y_direction
            err -= 1
```

三角形线框渲染



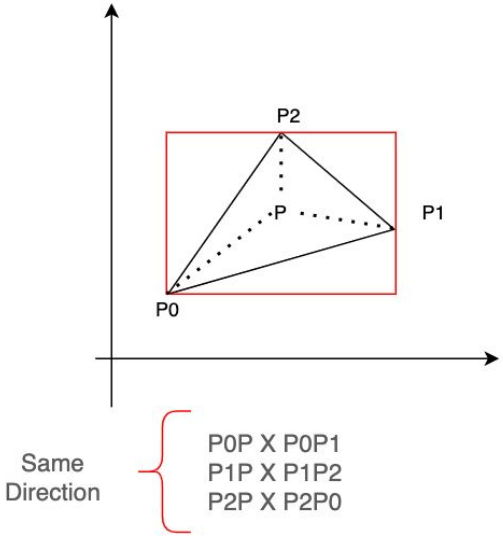
```
m = Model("obj/african_head.obj")

width = height = 800
image = CoordinateImage(width, height)
white = (255, 255, 255)

for face in m.faces():
    for n_edge in range(3):
        v0 = m.vert(face[n_edge])
        v1 = m.vert(face[(n_edge + 1) % 3])
        x0 = int((v0[0] + 1) * width / 2)
        y0 = int((v0[1] + 1) * height / 2)
        x1 = int((v1[0] + 1) * width / 2)
        y1 = int((v1[1] + 1) * height / 2)

        line(x0, y0, x1, y1, image, white)

image.save("african_head_wire")
```

三角形填充(线条扫描)




```python
if p0.y == p1.y == p2.y:
    return
# sort points by their y value
p0, p1, p2 = sorted([p0, p1, p2], key=lambda p: p.y)
total_h = p2.y - p0.y
for i in range(total_h):
    is_top_half = i > (p1.y - p0.y) or (p0.y == p1.y)
    segment_h = p2.y - p1.y if is_top_half else p1.y - p0.y
    alpha = i / total_h
    beta = ((i - (p1.y - p0.y)) if is_top_half else i) / segment_h
    A = p0 + (p2 - p0) * alpha
    B = p1 + (p2 - p1) * beta if is_top_half else p0 + (p1 - p0) * beta
    if A.x > B.x:
        A, B = B, A
    for x in range(A.x, B.x + 1):
        image.set(x, p0.y + i, color)
```

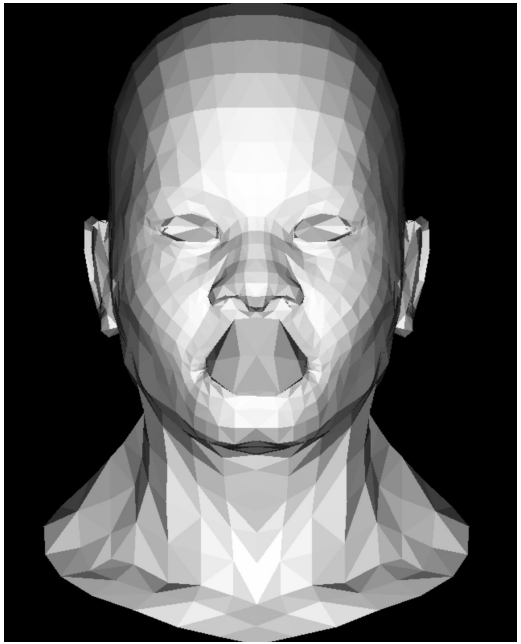
三角形填充(边界判定)




```
    # get the bounding box include the triangle
    min_x = min(p0.x, p1.x, p2.x)
    max_x = max(p0.x, p1.x, p2.x)
    min_y = min(p0.y, p1.y, p2.y)
    max_y = max(p0.y, p1.y, p2.y)

    for x in range(min_x, max_x + 1):
        for y in range(min_y, max_y + 1):
            if in_triangle_by_cross_pruduct(Vector2(x, y, p0.n_type), p0, p1, p2):
                image.set(x, y, color)
```

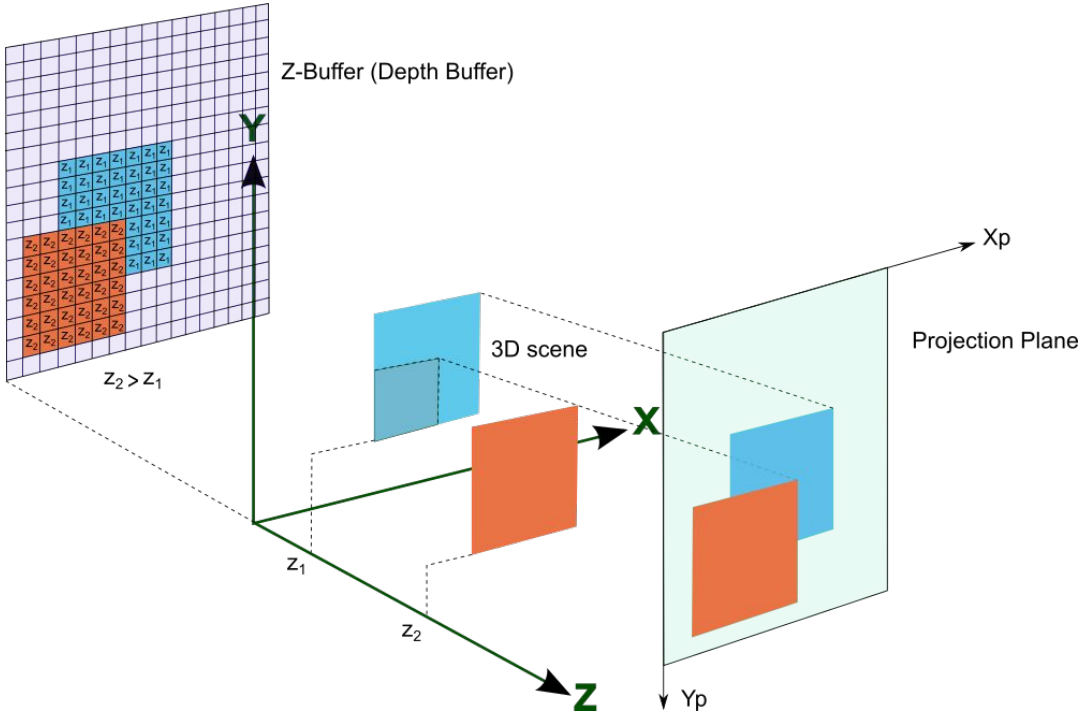
```
def in_triangle_by_cross_pruduct(p, p0, p1, p2):
    def z_direction(v1, v2, v3):
        l1 = v2 - v1
        l2 = v3 - v1
        return l1.x * l2.y - l1.y * l2.x >= 0

    d1 = z_direction(p, p0, p1)
    d2 = z_direction(p, p1, p2)
    d3 = z_direction(p, p2, p0)
    return d1 == d2 and d2 == d3
```

三角形填充效果

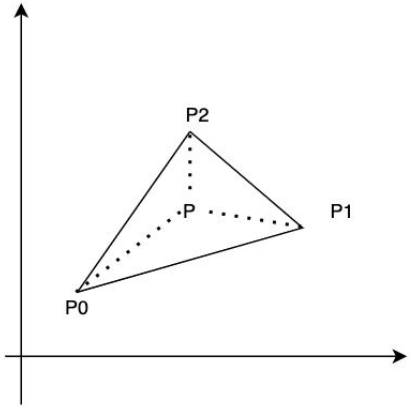


```python
m = Model("obj/african_head.obj")
width = height = 800
image = CoordinateImage(width, height)
white = (255, 255, 255)
light = Vector3(0, 0, -1, float)
for face in m.faces():
    screen_coords: typing.List[Vector2] = []
    world_coords: typing.List[Vector3] = []
    for n_edge in range(3):
        v0 = Vector3(*m.vert(face[n_edge]), float)
        screen_coords.append(
            Vector2((v0.x + 1) * width / 2, (v0.y + 1) * height / 2)
        )
        world_coords.append(v0)
    n = (world_coords[2] - world_coords[0]) ^ (world_coords[1] - world_coords[0])
    n.normalize()                                    光强度
    intensity = light * n
    if intensity > 0:
        triangle.triangle(
            screen_coords[0],
            screen_coords[1],
            screen_coords[2],
            image,
            (int(255 * intensity), int(255 * intensity), int(255 * intensity)),
        )
image.save("african_head_triangle")
```

Z-Buffer


Z-Buffer (Depth Buffer)
Y
$z_2 > z_1$
3D scene
X
Xp
Projection Plane
z_1
z_2
Z
Yp

Z-Buffer：三角重心坐标（Barycentric Cord）

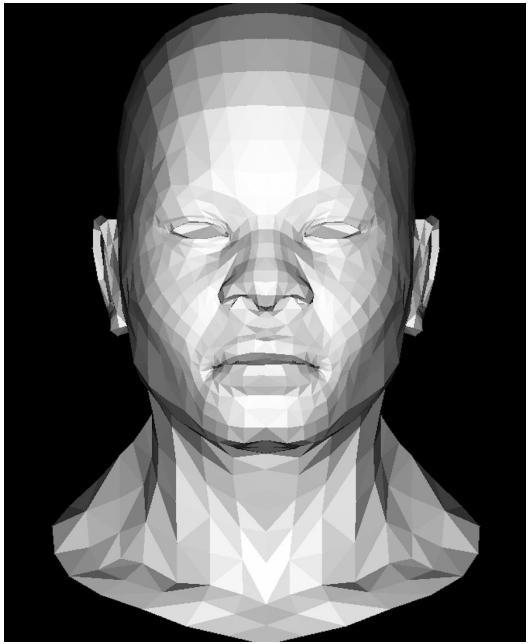


求 u, v, w 三个值，满足:

1. u + v + w = 1
2. u * P0 + v * P1 + w * P2 = P

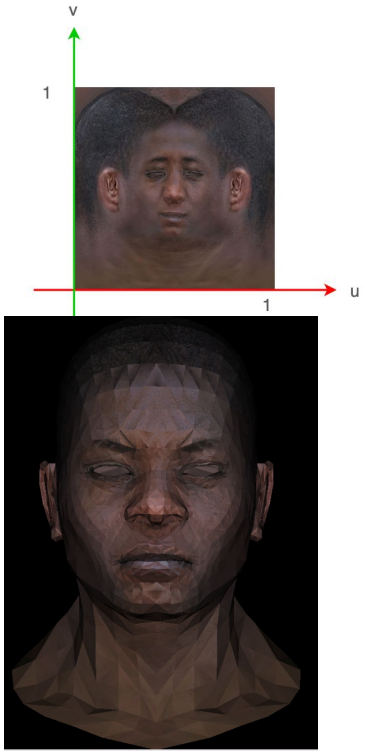
```
def barycentric(p, p0, p1, p2) -> Vector3:
    # AP = uAB + vAC  ==> P = (1 - u - v)A + uB + vC
    # return (1 - u - v, u, v)
    u = Vector3(p2.x - p0.x, p1.x - p0.x, p0.x - p.x, float) ^ Vector3(
        p2.y - p0.y, p1.y - p0.y, p0.y - p.y, float
    )
    if abs(u.z) < 0.01:
        return Vector3(-1, 1, 1, float)
    return Vector3(1 - ((u.x + u.y) / u.z), u.x / u.z, u.y / u.z, float)
```

Z-Buffer：消除隐藏背景



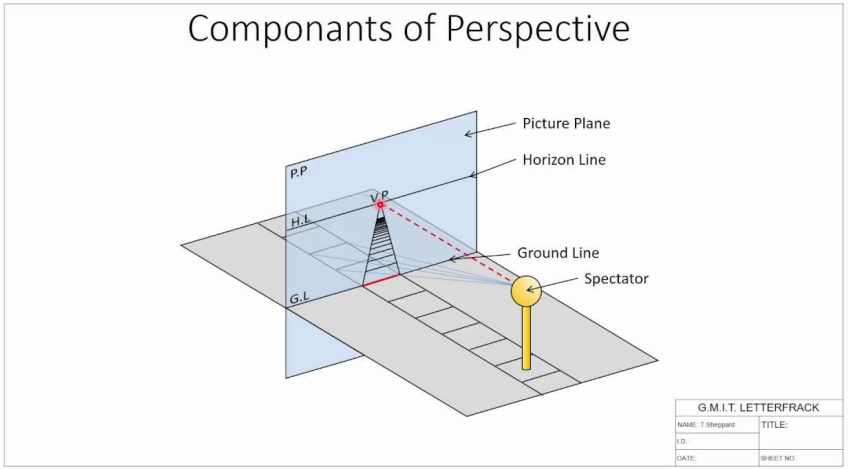
```
# get the bounding box include the triangle
min_x = int(min(p0.x, p1.x, p2.x))
max_x = int(max(p0.x, p1.x, p2.x))
min_y = int(min(p0.y, p1.y, p2.y))
max_y = int(max(p0.y, p1.y, p2.y))

for x in range(min_x, max_x + 1):
    for y in range(min_y, max_y + 1):
        bc_u = barycentric(
            Vector2(x, y),
            Vector2(p0.x, p0.y),
            Vector2(p1.x, p1.y),
            Vector2(p2.x, p2.y),
        )
        if bc_u.x < 0 or bc_u.y < 0 or bc_u.z < 0:
            continue
        z_value = Vector3(p0.z, p1.z, p2.z, float) * bc_u
        idx = x + y * width
        if z_buffer[idx] < z_value:
            z_buffer[idx] = z_value
            image.set(x, y, color)
```

纹理（Texture）




```
for x in range(min_x, max_x + 1):
    for y in range(min_y, max_y + 1):
        if x > width or y > height:
            continue
        bc_u = barycentric( Vector2(x, y), Vector2(p0.x, p0.y), Vector2(p1.x, p1.y), Vector2(p2.x, p2.y),)
        if bc_u.x < 0 or bc_u.y < 0 or bc_u.z < 0:
            continue
        z_value = Vector3(p0.z, p1.z, p2.z, float) * bc_u
        texture_x = int( ( texture_coords[0].x * bc_u.x + texture_coords[1].x * bc_u.y + texture_coords[2].x * bc_u.z) * texture_w)
        texture_y = int( ( texture_coords[0].y * bc_u.x + texture_coords[1].y * bc_u.y + texture_coords[2].y * bc_u.z) * texture_h)
        color = texture.get_pixel(texture_x, texture_y)
        idx = x + y * width
        if z_buffer[idx] < z_value:
            z_buffer[idx] = z_value
            try:
                image.set(x, y, [int(c * intensity) for c in color])
            except:
                print("x = ", x, "y = ", y)
                raise
```

透视投影（Perspective Projection）



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix}$$

The retro-projection gives us the following 3D coordinages:

$$\begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{rz+1} \\ \frac{y}{rz+1} \\ \frac{z}{rz+1} \end{bmatrix}$$

透视投影效果

```
m = Model1("obj/african_head.obj")
texture = CoordinateImage()
texture.load("obj/african_head_diffuse.tga")
width = height = 800
depth = 255
image = CoordinateImage(width, height)
light = Vector3(0, 0, -1, float)
camera = Vector3(0, 0, 3, float)    相机点
z_buffer = defaultdict(lambda: -1000)
projection = np.identity(4)
projection[3][2] = -1 / camera.z    透视转换矩阵
viewport = create_viewport(width, height, depth)
transfer_matrix = viewport.dot(projection)
for face in m.faces():
    screen_coords: typing.List[Vector3] = []
    world_coords: typing.List[Vector3] = []
    texture_coords: typing.List[Vector3] = []  # texture coords
    for idx in range(3):
        v0 = m.vert(face.vert_idx[idx])
        v0_t = transfer_matrix.dot(np.array([[v0.x], [v0.y], [v0.z], [1.0]]))
        screen_coords.append( Vector3( v0_t[0][0] / v0_t[3][0], v0_t[1][0] / v0_t[3][0], v0_t[2][0] / v0_t[3][0], float,))
        world_coords.append(v0)
        texture_coords.append(m.texture(face.texture_idx[idx]))

    n = (world_coords[2] - world_coords[0]) ^ (world_coords[1] - world_coords[0])
    n.normalize()
    intensity = light * n
    if intensity > 0:
        triangle.triange_with_texture( screen_coords, texture_coords, width, height, z_buffer, image, texture, intensity,)

image.save("african_head_triangle_with_perspective")
```

参考

- GAMES101: 现代计算机图形学入门 (https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html)
- Tiny renderer or how OpenGL works: software rendering in 500 lines of code (https://github.com/ssloy/tinyrenderer)

Q & A