Normal

1 用深度重建法线（Unity 的 URP：SSAO）

1.1 起因

（1）用 MRT 存储法线通道图，对移动端不太友好

（2）法线通道存储的可能是包含法线贴图的法线，对某些算法不友好；分别存储几何法线和着色法线，显存占用较大

1.2 Low-Level

利用偏导数计算法线，GPU 是用 2×2 的 quad 进行计算，可以计算局部的偏导数；但是，对于该 quad 的每个 pixel，法线是一致的，精度较低，如果处于边缘位置，会出现 artifact。

```
1 float3 normal = normalize(cross(ddy(viewPos), ddx(viewPos)));
```

1.3 Middle-Level

在水平方向上，左右各采样一个点；垂直方向上，上下各采样一个点；分别选择水平方向上和垂直方向上深度最大的点，重建观察空间的坐标，和当前像素点组成三角形的三个点，利用叉乘计算法线。

注：选择深度最大的点，是为了考虑物体边缘位置的情况，避免出现 artifact。

1.4 High-Level

相比于 middle，在水平方向上，左右各采样两个点；垂直方向上，上下各采样两个点。

1.5 参考

Accurate Normal Reconstruction from Depth Buffer
Improved normal reconstruction from depth

2 法线压缩技术

目标：在存储精度和存储占用之间进行平衡

2.1 Octahedron Environment Maps

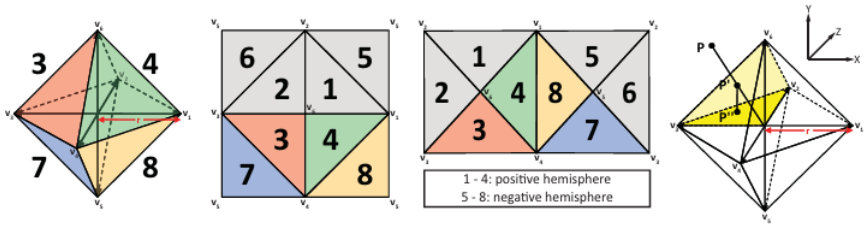

图 1: Octahedron Environment Maps

整体思想：将单位球上的向量投影到正八面体上，然后将正八面体展开成 2D 正方形

2.1.1 算法步骤

（1）将单位球上的点映射到正八面体上，记单位球上的点为 P，对应的正八面体上的点为 $P^{'}$

$$P^{'} = \frac{P}{|P_x| + |P_y| + |P_z|}$$

（2）如图1所示，竖直方向为 y 方向，计算 XZ 平面上的投影点

（2.1）计算 $P^{'}$ 在第二幅图里对应的坐标，取值范围为 $[-1, 1]^2$

$$P^{''} = \begin{cases} (P_x^{'}, 0, P_z^{'})^T & P_y^{'} \geq 0 \\ (sign(P_x^{'})(1 - sign(P_z^{'}))P_z^{'}, 0, sign(P_z^{'})(1 - sign(P_x^{'}))P_x^{'})^T & P_y^{'} < 0 \end{cases}$$

（2.2）将 P'' 映射到第三幅图里对应的坐标，取值范围为 $[-2,2]\times[-1,1]$

$$P'' = \begin{cases} (P'_x - P'_z - 1, 0, P'_x + P'_z)^T & P'_y \geq 0 \\ (P'_z - P'_x - 1, 0, P'_x + P'_z)^T & P'_y < 0 \end{cases}$$

此处贴一下 Unity 中的实现

```
// Ref: http://www.vis.uni-stuttgart.de/~engelhts/paper/vmvOctaMaps.pdf
// Encode with Oct, this function work with any size of output
// return real between [-1, 1]
real2 PackNormalOctRectEncode(real3 n)
{
    // Perform planar projection.
    real3 p = n * rcp(dot(abs(n), 1.0));
    real  x = p.x, y = p.y, z = p.z;

    // Unfold the octahedron.
    // Also correct the aspect ratio from 2:1 to 1:1.
    real r = saturate(0.5 - 0.5 * x + 0.5 * y);
    real g = x + y;

    // Negative hemisphere on the left, positive on the right.
    return real2(CopySign(r, z), g);
}

real3 UnpackNormalOctRectEncode(real2 f)
{
    real r = f.r, g = f.g;

    // Solve for {x, y, z} given {r, g}.
    real x = 0.5 + 0.5 * g - abs(r);
    real y = g - x;
    // EPS is absolutely crucial for anisotropy
    real z = max(1.0 - abs(x) - abs(y), REAL_EPS);

```

```
        real3 p = real3(x, y, CopySign(z, r));

        return normalize(p);
    }
```

参考：

Octahedron Environment Maps

2.2 Fast Oct Encode

2.2.1 论文中的实现版本

```
vec2 signNotZero(vec2 v) {
    return vec2((v.x >= 0.0) ? +1.0 :-1.0, (v.y >= 0.0) ? +1.0 :-1.0);
}

// Assume normalized input. Output is on [-1, 1] for each component.
vec2 float32x3_to_oct(in vec3 v) {
    // Project the sphere onto the octahedron, and then onto the xy plane
    vec2 p = v.xy * (1.0 / (abs(v.x) + abs(v.y) + abs(v.z)));
    // Reflect the folds of the lower hemisphere over the diagonals
    return (v.z <= 0.0) ? ((1.0- abs(p.yx)) * signNotZero(p)) : p;
}

vec3 oct_to_float32x3(vec2 e) {
    vec3 v = vec3(e.xy, 1.0- abs(e.x)- abs(e.y));
    if (v.z < 0) v.xy = (1.0- abs(v.yx)) * signNotZero(v.xy);
    return normalize(v);
}
```

2.2.2 Unity 中优化版本

```
float2 PackNormalOctQuadEncode(float3 n)
{
    n *= rcp(max(dot(abs(n), 1.0), 1e-6));
```

```
    float t = saturate(-n.z);
    return n.xy + float2(n.x >= 0.0 ? t : -t, n.y >= 0.0 ? t : -t);
}

float3 UnpackNormalOctQuadEncode(float2 f)
{
    // NOTE: Do NOT use abs() in this line.
    // It causes miscompilations. (UUM-62216, UUM-70600)
    float3 n = float3(f.x, f.y,
         1.0 - (f.x < 0 ? -f.x : f.x) - (f.y < 0 ? -f.y : f.y));

    float t = max(-n.z, 0.0);
    n.xy += float2(n.x >= 0.0 ? -t : t, n.y >= 0.0 ? -t : t);

    return normalize(n);
}
```

2.3 Unity 补充版本

```
real2 PackNormalHemiOctEncode(real3 n)
{
    real l1norm = dot(abs(n), 1.0);
    real2 res = n.xy * (1.0 / l1norm);

    return real2(res.x + res.y, res.x - res.y);
}

real3 UnpackNormalHemiOctEncode(real2 f)
{
    real2 val = real2(f.x + f.y, f.x - f.y) * 0.5;
    real3 n = real3(val, 1.0 - dot(abs(val), 1.0));

    return normalize(n);
```

```
15 }
```

参考：

A Survey of Efficient Representations for Independent Unit Vectors