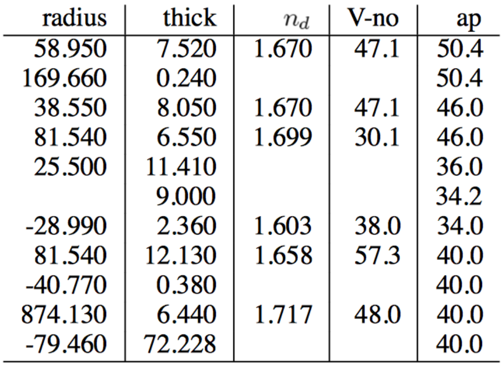
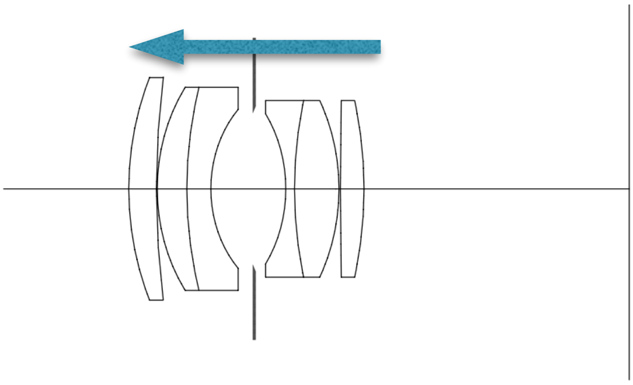
Rendering HW2

資工所 - R04922078 - 吳德彥

ParseLensData:



radius	thick	n_d	V-no	ap
58.950	7.520	1.670	47.1	50.4
169.660	0.240			50.4
38.550	8.050	1.670	47.1	46.0
81.540	6.550	1.699	30.1	46.0
25.500	11.410			36.0
	9.000			34.2
-28.990	2.360	1.603	38.0	34.0
81.540	12.130	1.658	57.3	40.0
-40.770	0.380			40.0
874.130	6.440	1.717	48.0	40.0
-79.460	72.228			40.0

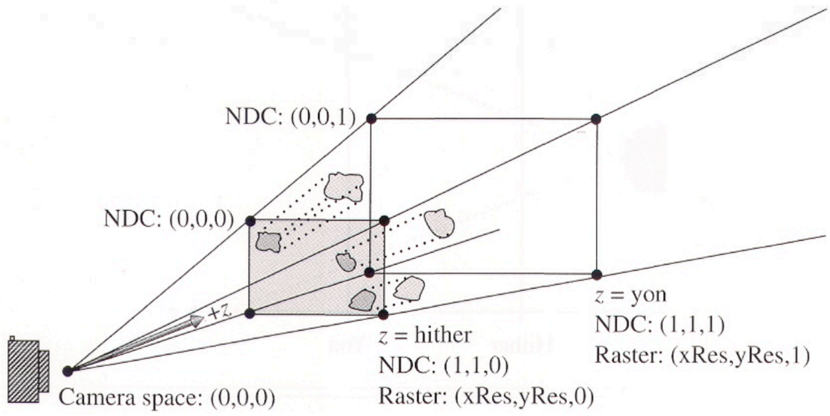


- 依據作業的data檔以及論文（*A Realistic Camera Model for Computer Graphics*） ，鏡片順序是右到左，且第一個鏡片位置位於原點(Camera Space)。
- 從data檔，可以得到radius, thick, nd, aperture
- 之後的每一個鏡片的位置是前面鏡片的位置減去前面鏡片的厚度(厚度=thick, sep, axpos)，程式如下：

lens[i].z = lens[i-1].z - lens[i-1].sep

- nd = 0 代表 Air 也是 Aperture stop，所以如果遇到nd=0，須先標記此鏡片是空氣也就是Aperture stop

RasterToCamera:

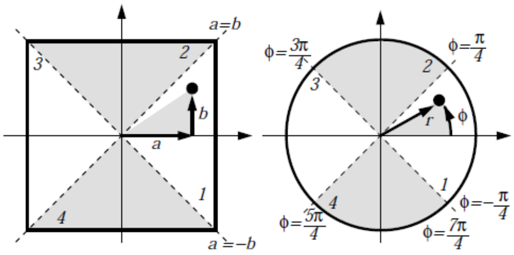


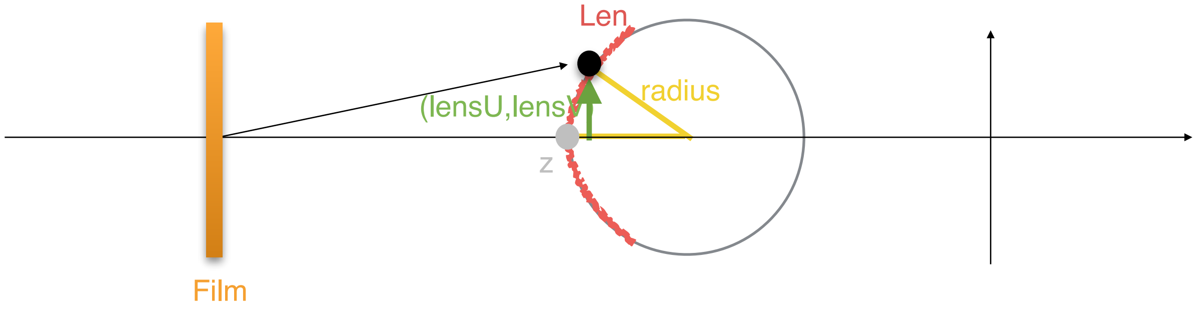
- Pbrt 的 filter sample是定義在 raster space底下，我們必須要轉回到camera space去做計算ray trace

- Raster -> NDC , 需要scale(ratio, ratio, 1),此階段只是把xRes, yRes,變成所小成單位唯一的空間
- NDC -> Camera, 需要 Translate(Vector(X, -Y, 0.f)) * Translate(Vector(0.f, 0.f, filmlocation)) * Scale(-1.f, 1.f, 1.f), 此階段先使中心點位移至Screen的中心，然後移動z軸至file的位置，接下來需要反轉x軸，以符合camera space

Sample Point:

採用內建函數 CencentricSampleDisk()，採用 A Low Distortion Map Between Disk and Square 論文中提到的方案，將一個正方形壓縮到一個圓形中。





lensZ= _len[lenIndex].z - radius + (radius < 0 ? -1 : 1) * (sqrt(radius*radius- lensU * lensU - lensV *

(radius < 0 ? -1 : 1) -> *The first column gives the signed radius of curvature of a spherical element; if none is given, the surface is planar. A positive radius of curvature indicates a surface that is convex when viewed from the front of the lens, while a negative radius of curvature is concave.(from A Realistic Camera Model for Computer Graphics)*

Sample Point -> (lensU, lensV, lensZ)

Ray Sphere Intersection:

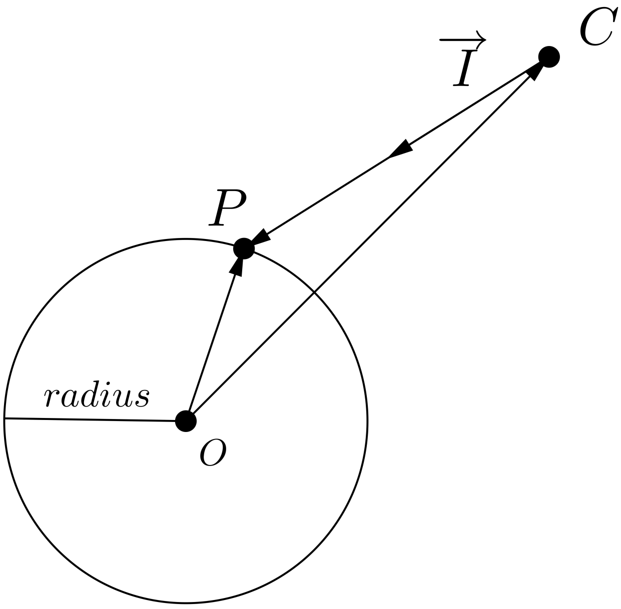
OC + I * t = OP

| OP | = radius

| OC + I* t | = | OP |

| OC + I * t | = radius

| OC | ^ 2 + 2 * (OC ・ I) * t + | I * t | ^ 2 - radius * radius = 0

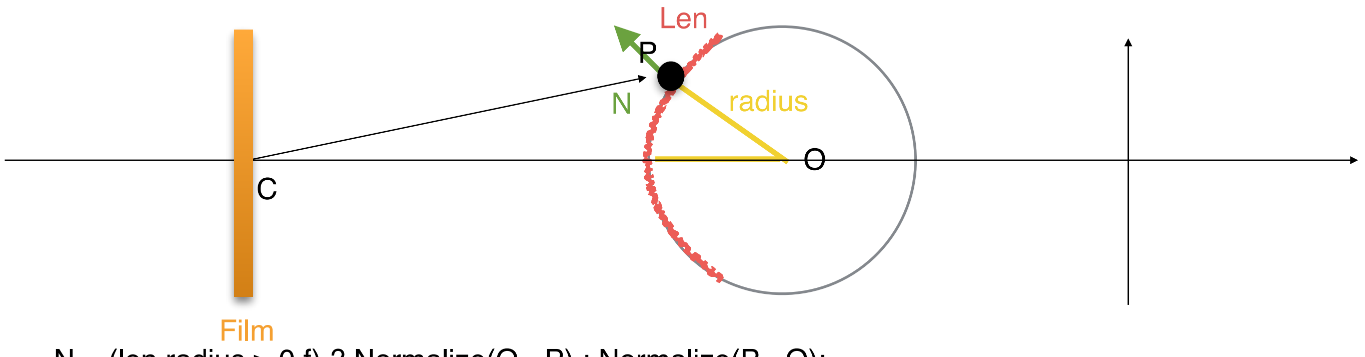


公式解法求 t ->

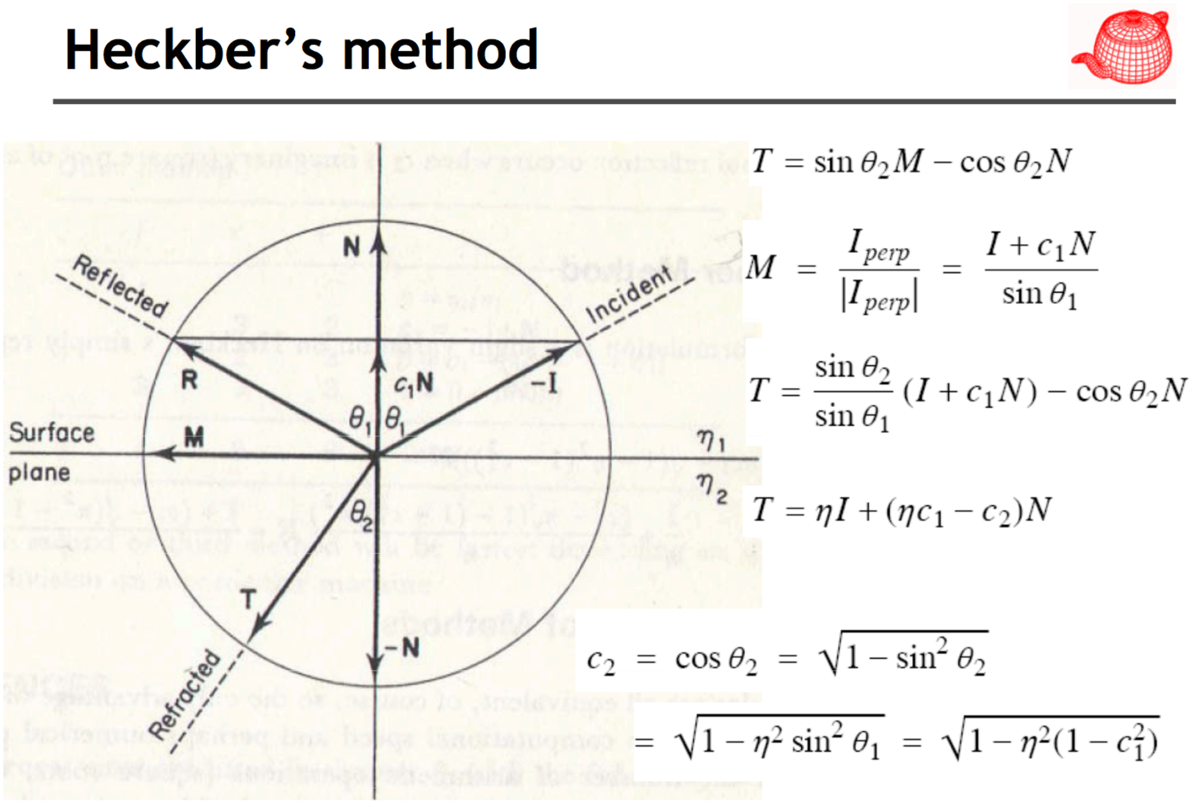
```
   float b = Dot(OC, I) * 2;
   float c = OC.LengthSquared() - surface.radius * surface.radius;
   float a = 1; // I.LengthSquared() = 1
   float determine = b * b - 4 * a * c;
   float t = 0;
   if (determine < 0) {
       return false;
   }
   else {
       float root = sqrtf(determine);
       t = (len.radius > 0) ? (-b + root) / (2 * a) : (-b - root) / (2 * a);
   }
```

Refraction：

求Normal:



```
N = (len.radius > 0.f) ? Normalize(O - P) : Normalize(P - O);
```



Heckber’s method

$$T = \sin\theta_2 M - \cos\theta_2 N$$

$$M = \frac{I_{perp}}{|I_{perp}|} = \frac{I + c_1 N}{\sin\theta_1}$$

$$T = \frac{\sin\theta_2}{\sin\theta_1}(I + c_1 N) - \cos\theta_2 N$$

$$T = \eta I + (\eta c_1 - c_2)N$$

$$c_2 = \cos\theta_2 = \sqrt{1 - \sin^2\theta_2}$$

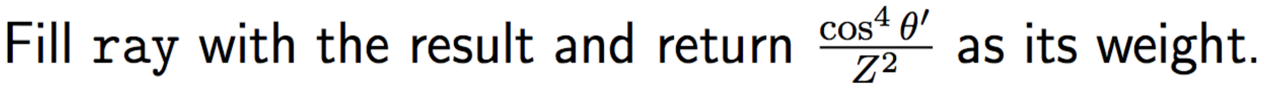
$$= \sqrt{1 - \eta^2 \sin^2\theta_1} = \sqrt{1 - \eta^2(1 - c_1^2)}$$

// Heckber's Method

```
  float n_ratio = len.n / n2;
  float c1 = -Dot(ray.d, N);
  float c2 = 1.f - n_ratio * n_ratio * (1.f - c1 * c1);
  if (c2 <= 0.f) return false;
  else c2 = sqrtf(c2);
  ray.d = Normalize(n_ratio * ray.d + (n_ratio * c1 - c2) * N);
```

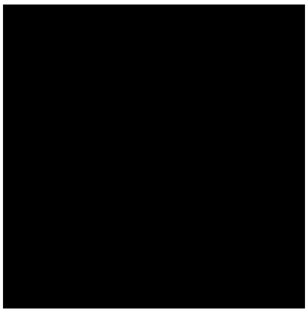
Ray Weight：

作業要求經過 float GenerateRay() 回傳



Fill ray with the result and return $\frac{\cos^4 \theta'}{Z^2}$ as its weight.

但回傳會暗到什麼都看不見：（以dof-dragons.dgauss.exr為例）

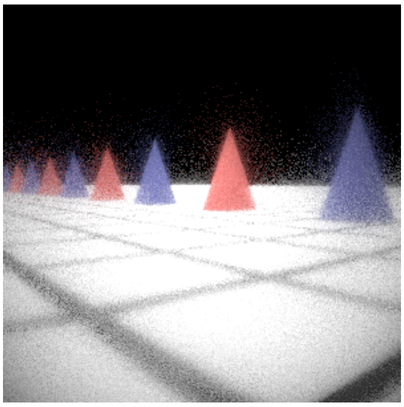


Refer to A Realistic Camera Model for Computer Graphics，multiply the result by the area of len:

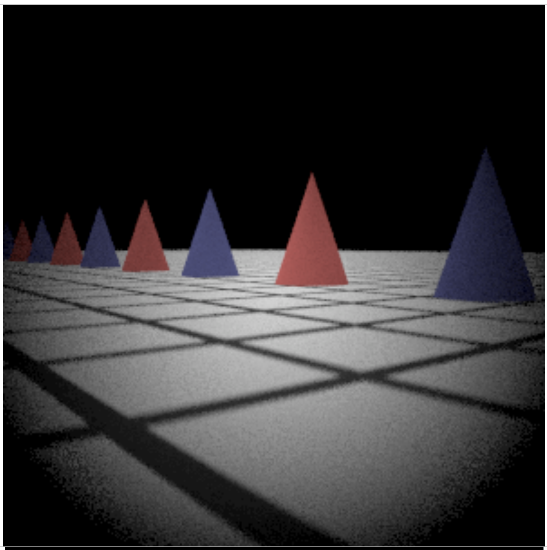
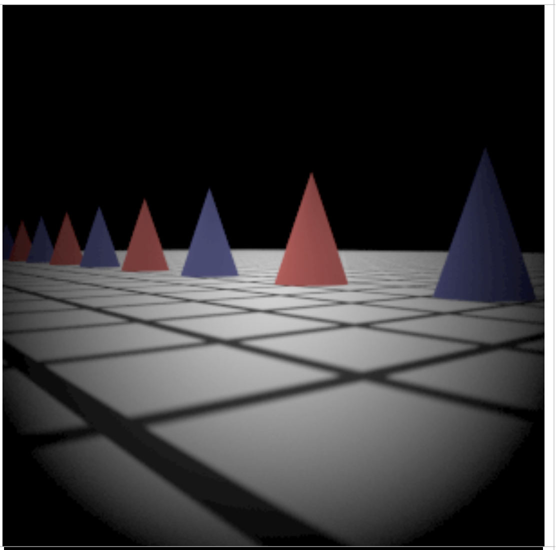
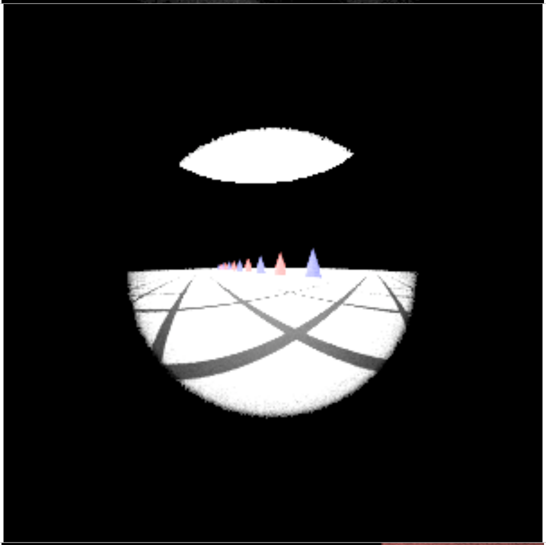
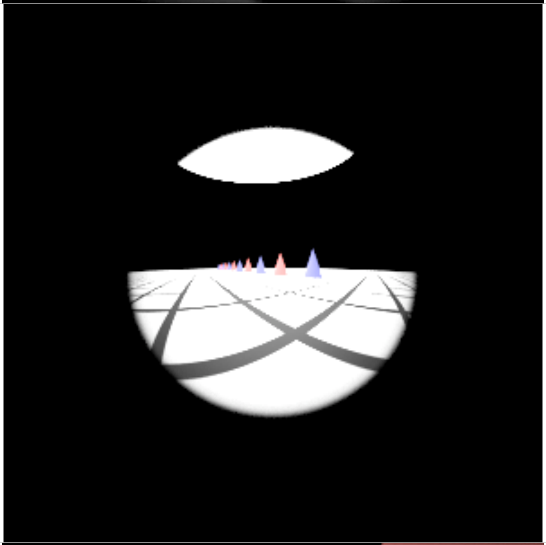
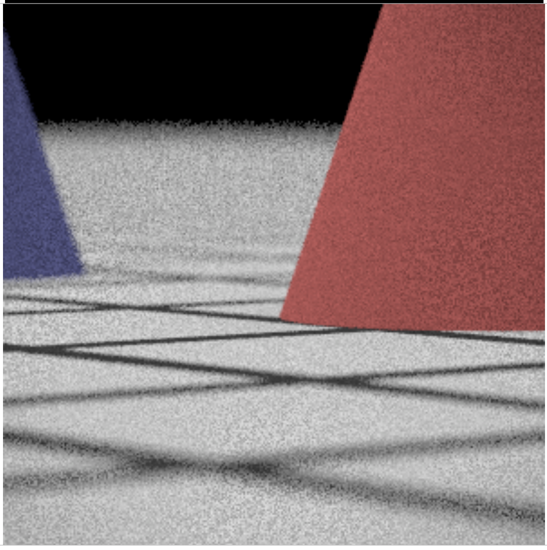
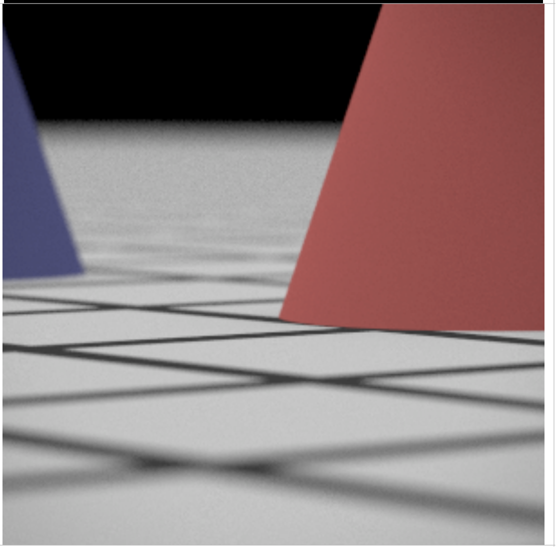
Code:

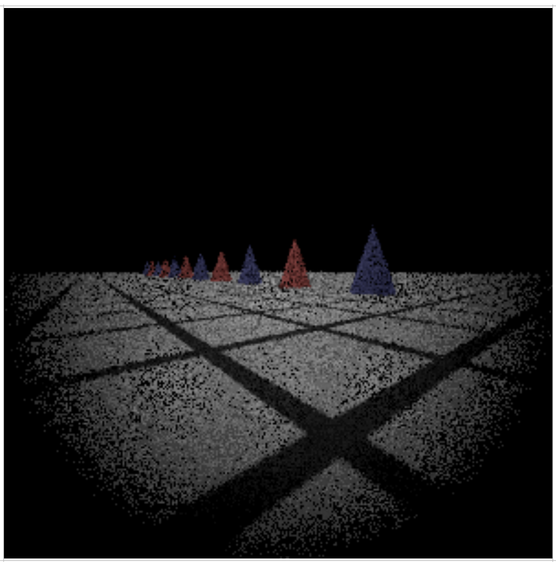
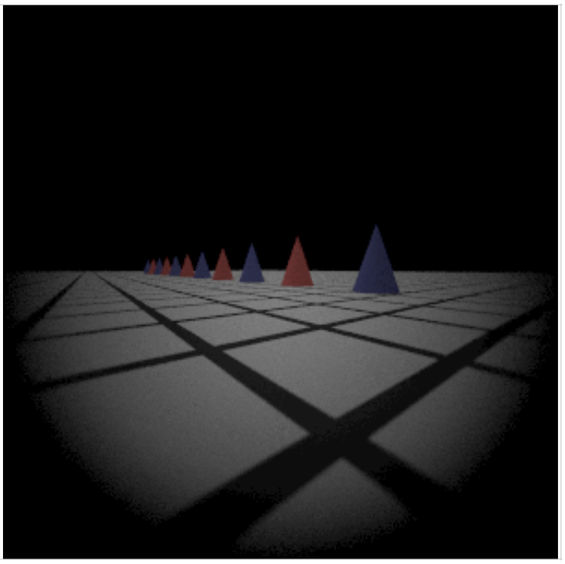
```
    // Set weight
    float cosTheta = Dot(Normalize(ray->o - cameraP), Vector(0, 0, 1));
    float z =  fabs(_location);
    float weight = (_lens[0].aperture * _lens[0].aperture * M_PI) / (z * z);

    weight = weight * cosTheta * cosTheta * cosTheta * cosTheta;

    CameraToWorld(*ray, ray);

    ray->d = Normalize(ray->d);

    return weight;
```

增加亮度後，結果會如下：（以dof-dragons.dgauss.exr為例）



所有結果圖：

	4 Sample	512 Samepl
dgauss.pbrt		
fisheye.pbrt		
telephoto.pbrt		

wide.pbrt		

執行環境(Mac.OS)及配置:

Core: 8 cores, 2.2Hz, I7
Memory: 16GB