

Homework 05

Scattering with 1-Simplexes

David C. Banks

Electrical Engineering and Computer Science
University of Tennessee

2009

Textbook

Chapter 4 Ray Tracing
Chapter 10 Surface Shading

Programming components

Write a program photon2d

A segment shoots photons into the plane

If a photon hits a segment, it may be absorbed

If a photon hits a segment, it may be scattered

A segment stores the scattering events

The scene is described in a text file

The scene is rendered as an image

Scene description

```
SCENE version 1.0  
dimension 2  
simplex1  
{  
simplex0 { 10.0 20.0 }  
simplex0 { 10.0 30.0 }  
emit 1500.0  
absorb 1.0  
}
```

Scene description

(continued)

```
simplex1
{
  simplex0 { 60.0 40.0 }
  simplex0 { 80.0 10.0 }
  emit 0.0
  absorb 0.2
}
```

1-sphere

Implement a 1-sphere data structure in \mathbb{R}^2

Center and radius: sph1.center, sph1.radius

Tangent direction: sph1.tangent

Normal direction: sph1.normal

Incident samples: sph1.in.numSamples, sph1.in.bin[0]

Exitant samples: sph1.out.numSamples, sph1.out.bin[0]

1-sphere

Tangent and normal define coordinate system for circle

Angle $\theta = 0$ along “tangent” direction

Angle $\theta = \pi/2$ along “normal” direction

Point (r, θ) in polar coordinates converts to

Point $(r \cos \theta, r \sin \theta)$ in tangent, normal coords

point.x = center.x + $r \cos \theta$ *tangent.x + $r \sin \theta$ *normal.x

point.y = center.y + $r \cos \theta$ *tangent.y + $r \sin \theta$ *normal.y

1-sphere samples

Sample a 1-sphere in \mathbb{R}^2
Hemicircle uniform distribution:
sphere1SampleHemiAngle(sph1)
angle = (drand48()-0.5)* π

1-sphere samples

Sample a 1-sphere in \mathbb{R}^2

Cosine distribution: `sphere1SampleCosineAngle(sph1)`

```
int accept = 0;
while (!accept)
{
    angle = (drand48()-0.5)* $\pi$ 
    accept = ( drand48() <= cos(angle) );
}
```

1-sphere angular density

Draw a starburst to represent $\rho(\theta)$ in polar coords

Segments emanate from circle center

The radius of each segment is $c_r * r * \rho(\theta)$

Value r = circle.radius

Number of angular samples = circle.numSamples

The gray level of each segment is $c_g * \rho(\theta)$

Pass scaleRadius c_r , scaleColor c_g from command line

1-sphere angular density

Segment from p_1 to p_2 on circle centered at p_1

Point $p_2 = (r, \theta)$ measured from p_1

$$p_2.x = p_1.x + r \cdot \cos \theta$$

$$p_2.y = p_1.y + r \cdot \sin \theta$$

1-sphere angles

Convert to vector from angle direction

`sphere1VectorFromAngle (vector, angle)`

`vector.x = cos(angle) * sph1.tangent;`

`vector.y = sin(angle) * sph1.tangent;`

Note: angle is measured from circle's "tangent"

1-sphere angles

Convert to angle on circle from vector direction

`sphere1AngleFromVector (sph1, θ , v)`

`vx = v · sph1.tangent;`

`vy = v · sph1.normal;`

`angle = atan2 (vy, vx)`

Note: angle is measured from circle's "tangent"

1-simplex

Implement a simplex1 data structure in \mathbb{R}^2
2 endpoints: `spx1.vertex[0]`, `spx1.vertex[1]`
Unit tangent in \mathbb{R}^2 : `spx1.tangent`
Unit normal in \mathbb{R}^2 : `spx1.normal`

1-simplex

Compute tangent from endpoints

$\text{tangent} = \text{spx1.vertex}[1] - \text{spx1.vertex}[0]$

$\text{tangentU} = \text{tangent} / \text{length}(\text{tangent})$

1-simplex

Compute normal from tangent

$$\text{normalU}.x = \text{tangentU}.y$$

$$\text{normalU}.y = -\text{tangentU}.x$$

1-simplex

Sample points on the 1-simplex

Array of bins: `spx1.inBin[0]`, `spx1.inBin[numBins-1]`

Each bin corresponds to an interval of the 1-simplex

Each interval has a circle with bins for in, out

`spx1.bin[0].sphere1.in.bin[0]`

`spx1.bin[0].sphere1.out.bin[0]`

1-simplex samples

Uniform random sampler for a segment

Barycentric coordinate $u \in [0, 1)$

```
u = drand48();
```

```
point = (1-u)*spx1->vertex[0] + u*spx1->vertex[1]
```

When $u=0$, point = $spx1 \rightarrow vertex[0]$

When $u=1$, point = $spx1 \rightarrow vertex[1]$

1-simplex coordinates

Convert from rectangular to barycentric coords

```
simplex1BaryFromRect ( spx1, u, point )
```

```
tangent = spx1->tangent; /* unit length */
```

```
delta = (point - spx1->vertex[0]);
```

```
u = dot(delta, tangent);
```

Point on line

```
Determine if point is on line through 1-simplex  
simplex1PointOnLine ( spx1, point )  
tangent = spx1->tangent; /* unit length */  
delta = (point - spx1->vertex[0]);  
u = dot(delta, tangent);  
if ( fabs(u) == length(delta) ) /* point is on line */
```

Point in simplex

Determine if point is in 1-simplex

`simplex1PointInside (spx1, point)`

First, find if point is on line through simplex

If so, find barycentric coordinate u of point

If u is in $[0,1]$, point is in simplex

Intersection

Determine segment-segment intersection

`intersectFlag = intersectSimplex1Simplex1 (point, spx1, spx2)`

If `flag == 1`, point is valid

Solve equations for 2 lines

See <http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/>

Intersection

Distance from point p in spx1 to q in spx2

$$pq = q - p$$

$$pqUnit = \frac{pq}{|pq|}$$

$$\text{Distance } u = pq \cdot pqUnit$$

if $u \leq 0$, point is behind simplex (discard it)

if $u < uMax$, q is the closest intersection so far

Intersection

```
foreach spx1 in simplexList where spx1.emit > 0
for ( photonNum=0; photonNum<spx1.emit; photonNum++ ) {
  p1, v1 = random position, direction on spx1;
  increment exitant bin for spx1;
  foreach spx2 in simplexList where spx2≠spx1 {
    find point of intersection between ray and spx2
    update the closest point in direction of ray
  }
  increment incident bin for closest point
  if (!absorbed)
  compute scatter direction, increment exitant bin
}
```

Program behavior

Read the scene containing segments

Part 1. For each emitter, sample points and directions

Part 1a. Create vector for each point, direction

Part 1b. Create starburst for each bin on the emitter

Program behavior

Part 2. Intersect point, direction with scene

Part 2a. Show intersection as little disk

Part 2b. Accumulate directions in incident bins

Part 2c. Create starburst for each incident bin

Program behavior

Part 3. Scatter

Part 3a. Create vector for each scatter event

Part 3b. Accumulate directions in exitant bins

Part 3c. Create starburst for each exitant bin