Visualização e Iluminação II - Projectos OSPray

Luís Paulo Santos – Dep. de Informática

Universidade do Minho

Maio, 2021

Descrevem-se abaixo os projectos propostos para Visualização e Iluminação II, ano lectivo 2020/21. Todos estes projectos implicam a utilização do renderer OSPRay da Intel.

Os projectos devem ser desenvolvidos em grupos de até 3 alunos. Como resultados finais a entregar incluem-se: uma apresentação (a realizar também oralmente em data a combinar), um relatório escrito e o produto do próprio projecto (código, modelos, etc.).

Lista de projectos:

1. **Elevado número de fontes de luz**
2. **Rendering selectivo**

Cronograma

24.Mai: Selecção do projecto

21..26.Jun: apresentação final e entrega relatório (15 minutos/grupo)

Descrição dos projectos

1. **Elevado número de fontes de luz**

Numa cena com um número elevado de fontes de luz o tempo dedicado ao processamento dos *shadow rays* rapidamente se transforma no custo mais significativo do *renderer*. Pretende-se comparar alguns métodos baseados em integração de Monte Carlo que seleccionam de forma inteligente qual a fonte de luz a amostrar (por exemplo, considerando as potências, áreas e distâncias relativas das várias fontes de luz).

Considere o código de `direct_illumination()` em `MyRenderer-T4.ispc`. É enviado um *shadow ray* para cada fonte de luz, fazendo com que o tempo de *rendering* de cada *frame* seja directamente proporcional ao número de fontes de luz, o que é claramente inaceitável.

Modifique o código da função `defineLightsCornell()` em `Renderer.cpp`, de forma a que crie um elevado número de fontes de luz (ex.: 36).

Crie em `MyRenderer-T4.ispc` o metodo `direct_illumination_1L()`. Deve seleccionar estocasticamente uma fonte de luz entre as existentes (são `Nlights = self->numLights` fontes de luz). Apenas será disparado um *shadow ray* na direcção da fonte de luz seleccionada. Uma vez que é amostrada apenas esta fonte de luz (em vez de amostrar todas) a contribuição desta fonte deve ser dividida pela probabilidade com que foi escolhida. A escolha foi feita usando uma distribuição uniforme, isto é, todas

as fontes de luz têm a mesma probabilidade, sendo esta igual ao inverso do número de fontes de luz existentes: `lightPDF = 1.f / self->numLights`. Note que a contribuição da fonte de luz selecionada tem que ser dividida por esta probabilidade:

```
color = color / lightPDF
```

Avalie o nível de ruído nas imagens geradas e a velocidade de convergência para uma imagem com pouco ruído.

Proponha formas mais eficazes de seleccionar a fonte de luz.

Sugestão: Se o método

```
Light_SampleRes light = l->sample(l, dg, s);
```

For chamado para todas as fontes de luz, as respectivas `Light_SampleRes` contêm no campo `light.weight` a potencial contribuição de cada uma das fontes de luz para a iluminação do ponto que está a ser processado (Note que é potencial porque a visibilidade não é conhecida: aquela fonte de luz pode estar ocludida por algum objecto e, nesse caso a sua contribuição será 0).

Podemos normalizar estas potências dividindo-as pela soma da potência total; chamamos a este vector a `pdf` e dá-nos a probabilidade de seleccionar cada uma das fontes de luz, dando maior probabilidade às fontes de luz com maior probabilidade de contribuir.

Para fazer esta selecção devemos calcular a probabilidade acumulada: `cdf`. Agora, sempre que quisermos seleccionar uma fonte de luz, obtemos um número aleatório entre 0 e 1, verificamos qual o 1º elemento da `cdf` que é maior ou igual que este número e usamos essa fonte de luz.

2. **Rendering selectivo**

Modifique o *renderer* fornecido com o Tutorial 4 para que para diferentes pixéis possam ser usados calculados usando diferentes parâmetros ou algoritmos.

O parâmetro `ScreenSample &sample` da função `MyRenderer_renderSample_RT()` permite identificar qual o pixel que está a ser processado: `(sample.sampleID.x, sample.sampleID.y)`. O parâmetro `FrameBuffer *uniform fb` indica a resolução da imagem: `(fb->size.x, fb->size.y)`.

Use esta funcionalidade para sintetizar diferentes regiões da imagem usando diferentes parâmetros. Exemplo: o código abaixo fixa todos os pixéis com da metade esquerda da imagem com a cor vermelha:

```
vec3f color;
if (sample.sampleID.x < fb->size.x/2)
  color = make_vec3f(1.f, 0.f, 0.f);
else
  color = recursive_PT (self, model, rng, ray, 0);
```

Aumentando aos parâmetros de `recursive_PT()` imagine formas de diferentes regiões da imagem serem processadas usando diferentes parâmetros.