

Ray-Tracing e Rasterização Modelação e Visualização

pg25333 César Perdigão
a61009 Luís Caseiro

13 de Fevereiro de 2014

Resumo

Com este trabalho é pretendido criar um *pipeline* gráfico híbrido usando rasterização e *Ray-Tracing*, usando a rasterização para a geração da imagem base e *Ray-Tracing* para produzir diversos efeitos como sombras, reflexões, refrações e oclusão ambiental. Pretende-se também medir o impacto de cada um destes efeitos, e a sua combinação, no desempenho e na qualidade da imagem final produzida.

Conteúdo

1	Introdução	4
2	Método	4
3	Rasterização	5
4	Sombras	5
5	Reflexões	7
6	Refracção	9
7	Oclusão Ambiental	11
8	Combinação dos Vários Efeitos	13
9	Conclusões	15
A	Ficheiros XML	18
A.1	SceneNoEffects.xml	18
A.2	SceneDummy.xml	20
A.3	SceneShadows.xml	26
A.4	SceneReflect.xml	31
A.5	SceneRefract.xml	37
A.6	SceneSSAO.xml	44
A.7	SceneFinal.xml	49
B	Ficheiros Cuda	56
B.1	deferredShadows.cu	56
B.2	reflection.cu	59
B.3	refraction.cu	64
B.4	ssao.cu	69
B.5	final.cu	73

Lista de Figuras

1	Imagen original sem nenhum efeito aplicado.	5
2	Diferença entre sombras resultantes de <i>Shadow Map</i> e <i>Ray-tracing</i>	5
3	Algoritmo de sombras com <i>Ray-Tracing</i>	6
4	Imagen original com sombras.	6
5	Reflexão com <i>Cube-Mapping</i>	7
6	Descrição de reflexão nos vidros.	7
7	Imagen original com reflexão nos vidros.	8
8	Imagen detalhada de reflexão no vidro.	8
9	Descrição de refração nos vidros.	9
10	Imagen original com reflexão e refração nos vidros.	10
11	Imagen detalhada de com reflexão e refração no vidro.	10
12	Erro na oclusão ambiental por rasterização.	11
13	Diagrama que descreve oclusão ambiental.	11

14	Imagen original aplicando oclusão ambiental com 16 amostras. . .	12
15	Imagen original aplicando oclusão ambiental com 49 amostras. . .	12
16	Imagen original aplicando oclusão ambiental com 100 amostras. .	13
17	Imagen final aplicando todos os efeitos.	14
18	Detalhe da imagem onde se verificam todos os efeitos.	14
19	Detalhe da imagem onde se verificam os efeitos de sombra, refração e reflexão.	15

1 Introdução

Existem dois grandes processos de gerar imagens através de computação, a rasterização e *Ray-Tracing*.

A rasterização é uma tecnica de produção de imagem em tempo real que se baseia na interpolação de posições e outros atributos de modelos poligonais para calcular a cor em cada pixel. É um método bastante eficiente e as arquiteturas das *GPU's* estão optimizadas para acelerar ao máximo este processo. Porem, não é possível obter imagens com o realismo desejado, dado que para acelerar o processo são realizadas algumas aproximações, e tem que se garantir que cada vertice dos modelos e cada pixel da imagem resultante é processado independentemente de todos os outros. Isto torna a implementação de certos efeitos de iluminação, como as sombras, imprecisos.

O *Ray-Tracing* baseia-se na simulação do trajecto que os raios de luz fazem no mundo real mas em sentido inverso, isto porque no mundo real uma fonte de luz emite raios de luz que irão percorrer o espaço até encontrar um objeto, estes raios irão sofrer alterações devido a fenomenos de reflexão e refração, por isso apenas uma grande minoria dos raios emitidos irão atingir os olhos do observador. Devido aos acontecimentos referidos optou-se por emitir raios a partir do observador de forma a processar apenas raios que são vistos por este. *Ray-Tracing* consegue produzir imagens com um nível alto de realismo, mas tem também um grande custo computacional, ou seja, para aplicações que corram em tempo real *Ray-Tracing* será pouco eficiente pois a velocidade de execução é fundamental.

O objetivo deste trabalho é tentar combinar o melhor dos dois métodos num *pipeline* híbrido, usando a rasterização para a produção da imagem base, e *Ray-Tracing* para a produção de vários efeitos de iluminação. Os efeitos que serão implementados serão sombras, reflexões, refrações e oclusão ambiental. Para cada um destes efeitos, e para a sua combinação, será medido o impacto no desempenho de forma a avaliar se é deseável a sua implementação em *Ray-Tracing*.

Para isto será usado *OpenGL* como interface de rasterização, e o motor de *Ray-Tracing Optix* da *Nvidia*, devido a facilidade de interoperabilidade com *OpenGL*, e a sua utilização já estar implementada no motor gráfico utilizado, o *Curitiba*.

2 Método

Para efetuar a medição de desempenho, será usado o modelo do Largo de Camões. Estas medições serão feitas num computador portátil *ASUS G750JH*, com o processador *Intel core i7 4700HQ*, 32 GB de memória RAM, e a placa gráfica *Nvidia Gefore GTX780M*. O sistema operativo utilizado é o *Windows 8.1* e o compilador *Visual C++ Compiler 2012*. Todos os *drivers* estavam atualizados à data de realização deste relatório. Foi usada a versão 3.0.1 do motor *Optix*. Para todas as cenas será usada a resolução de 1024 por 1024 pixels.

3 Rasterização

Como ponto de partida, temos uma imagem gerada usando apenas rasterização Figura 1, sem qualquer efeito aplicado, usando apenas o modelo de iluminação Phong.



Figura 1: Imagem original sem nenhum efeito aplicado.

Para esta cena, o desempenho obtido foi de 1100 FPS. Este será o valor de referência para avaliar o custo de implementação de cada um dos efeitos.

Para melhor ter uma noção do impacto dos efeitos utilizados, foi criado um passo em que o *Optix* não faz nada para além de devolver como resultado o valor de cor da entrada. Neste exemplo foi obtido um desempenho de 200 FPS, uma quebra desempenho de 82% em relação ao original.

4 Sombras

Seria desejável uma implementação de sombras usando *Ray-Tracing* devido aos erros comuns que aparecem nas sombras em rasterização usando *Shadow Maps*, como podemos ver na Figura 2, e a possibilidade de criar *Soft Shadows*, difícil de implementar usando *Shadow Volumes*.

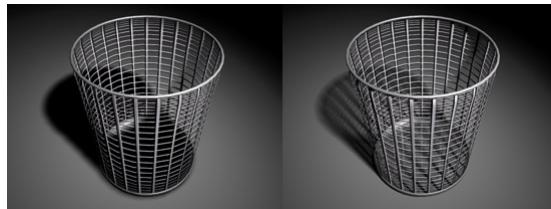


Figura 2: Diferença entre sombras resultantes de *Shadow Map* e *Ray-tracing*.

O cálculo das sombras usando *Ray-Tracing* é bastante simples, bastando apenas traçar um raio na direção da luz, e caso este intercepte algum objeto, o ponto está em sombra. Caso contrário, está iluminado. Como temos a imagem rasterizada como ponto de partida, as posições de partida de cada raio já são conhecidas para cada pixel. Tendo isto, o custo global do cálculo das sombras é de um raio traçado por cada pixel.

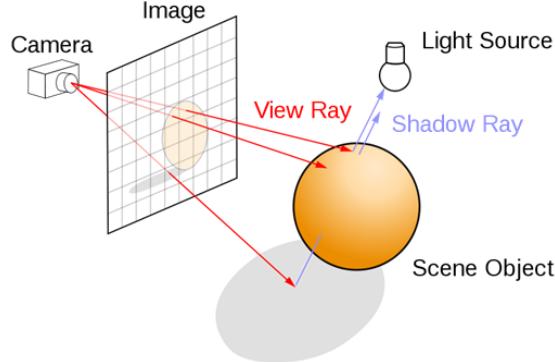


Figura 3: Algoritmo de sombras com *Ray-Tracing*.

Uma optimização implementada para diminuir o número de raios traçados é apenas traçá-los quando os objectos estão voltados para a luz, dado que já é possuída essa informação resultante da rasterização.

Na Figura 4 podemos ver o resultado da aplicação de sombras à imagem original.



Figura 4: Imagem original com sombras.

Para esta cena, o desempenho obtido foi de 90 FPS, uma perda de cerca de 92% em relação à cena original, e de 55% em relação ao passo com *Optix* sem efeitos. Apesar do custo elevado, ainda são produzidas imagens em tempo real.

5 Reflexões

A implementação de reflexões em rasterização usando *Cube Map* tem algumas falhas, nomeadamente a ausência de auto-reflexão, como podemos ver na Figura 5. Para além disso, apesar das direções de reflexão serem corretas, o ponto de foco da reflexão é sempre considerado o centro do modelo, e não o ponto atingido pela luz. Postos estes inconvenientes, seria desejável implementar reflexões mais precisas usando *Ray-Tracing*.



Figura 5: Reflexão com *Cube-Mapping*.

Para calcular reflexões usando *Ray-Tracing* temos de calcular o vector que vai da camera para o objecto e reflecti-lo segundo a normal do objecto. De seguida é traçado um raio segundo essa direção, que vai ter como resultado a cor do objecto mais próximo. Este raio pode por sua vez ser reflectido, o que dá origem a um processo recursivo. Este processo é parado caso a profundidade (número de reflexões) ultrapasse um dado valor. Para este exemplo, apenas objectos de vidro têm a capacidade de reflectir raios.

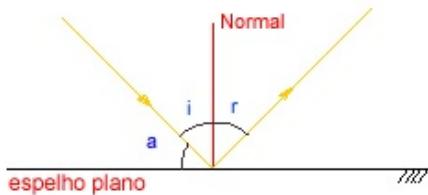


Figura 6: Descrição de reflexão nos vidros.

A complexidade deste algoritmo pode variar entre zero raios por pixel caso não haja vidro visível, até ao valor da profundidade máxima permitida. Dada a

orientação dos vidros na cena, não é de esperar que ocorram mais que 3 reflexões por raio.

Podemos ver na Figura 7 o resultado da aplicação de reflexões na cena original.



Figura 7: Imagem original com reflexão nos vidros.

O desempenho obtido foi de 140 FPS, provavelmente porque é visível pouco vidro na cena. Na Figura 8 podemos ver um detalhe da reflexão.



Figura 8: Imagem detalhada de reflexão no vidro.

Na segunda imagem o desempenho já é um pouco mais baixo, isto porque o vidro cobre uma maior parte da cena, caindo para 100 FPS. Isto corresponde a uma perda de desempenho de 90% em relação ao original, e de 50% em relação ao passo com *Optix* sem efeitos.

6 Refração

A refração é um fenômeno ótico que consiste no desvio de um feixe luminoso quando muda para um meio com diferente densidade. Esta defleção é dada pela equação:

$$\frac{\sin(\theta_i)}{\sin(\theta_t)} = \frac{n_2}{n_1} \quad (1)$$

em que θ_i é o ângulo de incidencia e θ_t é o ângulo de refração. $\frac{n_2}{n_1}$ é o índice de refração do material em relação ao do ar. Devido a na realidade um feixe de luz ser refratado duas vezes, uma ao entrar e outra ao sair do vidro, e o vidro ser representado apenas como um plano, o índice usado vai ser mais baixo que o índice real.

Os defeitos na refração por rasterização são essencialmente os mesmos que na reflexão, daí ser também desejável a sua implementação usando *Ray-Tracing*.

Para efectuar este efeito basta aplicar a equação 1 no raio incidente e calcular o raio refratado. Este raio pode no entanto não existir, pois para um ângulo crítico deixa de existir refração e existe apenas reflexão total.

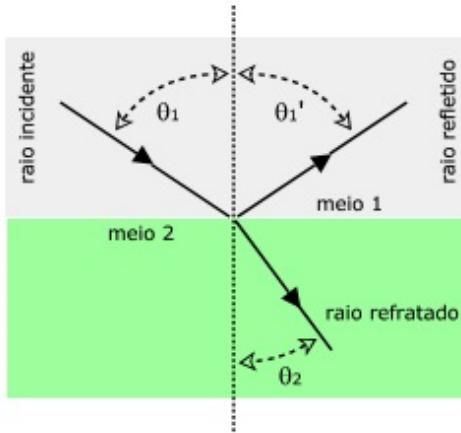


Figura 9: Descrição de refração nos vidros.

Tendo agora as direções dos raios refletido e refratado, estes raios são traçados e obtem-se a cor de cada um deles, sendo necessário agora atribuir pesos a cada uma das cores obtidas. Estes pesos são calculados por uma aproximação a uma das equações de Fresnell:

$$coef = 1 - K^2 \times (1 - dot(norm, dir)^2) \quad (2)$$

em que $coef$ é o coeficiente de luz refratada (sendo $1 - coef$ o coeficiente de luz refletida), $norm$ a normal à superfície e dir o vetor incidente. K é o índice de refração utilizado. Esta aproximação requer que ambos os vetores estejam normalizados. No fim basta multiplicar $coef$ pela cor refratada e $1 - coef$ pela cor refletida para obter a cor final. Caso não haja refração, a cor final é apenas a cor refletida.

A complexidade deste efeito pode chegar a 2^D (em que D é a profundidade máxima) raios por pixel, caso a cena envolva muitas reflexões e refrações. O resultado desta implementação é visível na Figura 10.



Figura 10: Imagem original com reflexão e refração nos vidros.

Na Figura 11 podemos ver um detalhe da cena com reflexões e refrações.



Figura 11: Imagem detalhada de com reflexão e refração no vidro.

O desempenho da aplicação nesta cena varia entre as 80 e 120 FPS, dependendo da quantidade de vidro visível na cena. Isto corresponde a um custo de 89% a 93% em relação à aplicação que só usa rasterização, ou 40% a 60% em relação à aplicação com *Optix*.

7 Oclusão Ambiental

A oclusão ambiental tenta simular o efeito de iluminação global, tentando representar o quanto um ponto está exposto a luz indireta.

A implementação em rasterização deste efeito usa a informação das profundidades e das normais de forma a avaliar a exposição de um determinado ponto. Contudo, só com esta informação, é possível que surjam erros quando a diferença de profundidades é muito grande, e na verdade os objetos mais próximos não encobrem os pontos mais distantes, como é possível ver na Figura 12.



Figura 12: Erro na oclusão ambiental por rasterização.

É ainda possível tentar corrigir este erro limitando a oclusão a uma diferença máxima de profundidade. Contudo, caso exista algum objeto entre o modelo mais próximo e o mais distante que cubra o último, uma zona é marcada como exposta quando não o está.

Em *Ray-Tracing*, este efeito é produzido traçando raios numa hemisfera em torno da normal do ponto, e contar quantos raios interceptam algum objeto. Quantos mais raios atingem a restante geometria, menos exposto esse ponto está, ficando visivelmente mais escuro. Para não ser visível o padrão na amostragem, é necessário aplicar algum ruído nos vetores a traçar, ficando o efeito mais uniforme.

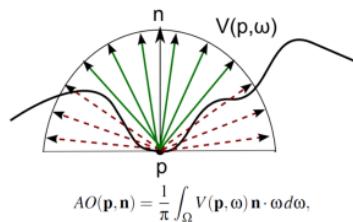


Figura 13: Diagrama que descreve oclusão ambiental.

Como é de esperar o número de raios traçados influenciará a qualidade da imagem e o desempenho da aplicação, como podemos ver de seguida.



Figura 14: Imagem original aplicando oclusão ambiental com 16 amostras.



Figura 15: Imagem original aplicando oclusão ambiental com 49 amostras.



Figura 16: Imagem original aplicando oclusão ambiental com 100 amostras.

Como é possível ver, só a partir das 49 amostras se começa a obter um resultado sem o efeito granulado, e mais preciso. Na Tabela 1 estão os resultados de desempenho para diferentes números de amostras por pixel.

Amostras	Tempo médio por frame (ms)
1	11.68
4	33.18
9	66.05
16	109.20
25	165.37
36	231.86
49	309.13
64	397.88
81	497.68
100	607.87

Tabela 1: Comparação entre número de amostras e respetivo desempenho.

Como é possível observar o impacto da oclusão ambiental é muito elevado, deixando de ser gerado em tempo real para imagens de maior qualidade. Pensou-se ainda filtrar o resultado da oclusão aplicando um esbatimento, de forma a diminuir o efeito granulado nas imagens com menos amostras. Contudo as imagens resultantes tinham sombras muito pouco definidas, sendo por isso omitidas.

8 Combinação dos Vários Efeitos

Como último teste, implementou-se uma combinação dos vários efeitos. Para além do cálculo normal das sombras e da oclusão ambiental para os pixels da imagem original, este cálculo tem agora também de ser feito para cada raio

refletido e refratado, sendo que o número de raios por pixel pode ser no maximo $2^D \times (OCC + 1) + OCC + 1$, em que OCC é o númeroso de amostras de oclusão ambiental e D é a profundidade máxima das reflexões e refrações. Podemos ver na Figura 17 o resultado final desta combinação.



Figura 17: Imagem final aplicando todos os efeitos.

Na Figura 18 é possivel observar em detalhe todos os efeitos, bem como a sua propagação pelos raios refetidos e refratados.



Figura 18: Detalhe da imagem onde se verificam todos os efeitos.

Como era expectável, o desempenho da aplicação foi baixo, ficando-se pelos 2 FPS. O principal inibidor de desempenho como visto anteriormente é a oclusão

ambiental, por isso foi feito um teste sem oclusão ambiental, que é visível na Figura 19.



Figura 19: Detalhe da imagem onde se verificam os efeitos de sombra, reflexão e refração.

Para este teste o desempenho foi de 60 FPS

9 Conclusões

Na Tabela 2 temos em detalhe a performance de cada um dos exemplos implementados, bem como a percentagem desse tempo que é gasto no *Optix*.

Efeito	Local	Tempo por frame (ms)	Percentagem do <i>Optix</i>
Sem efeitos	Fonte	0.87	
Sem efeitos com <i>optix</i>	Fonte	5.33	81
Sombras	Fonte	11.77	93,7
Reflexão	Fonte	7.16	86
Reflexão	Vidro	15.61	95,3
Refração	Fonte	9.4	89,1
Refração	Vidro	23.42	96,8
Oclusão Ambiental com $x = 1$	Fonte	11.68	91,8
Oclusão Ambiental com $x = 4$	Fonte	33.18	96,8
Oclusão Ambiental com $x = 9$	Fonte	66.05	98,3
Oclusão Ambiental com $x = 16$	Fonte	109.20	98,9
Oclusão Ambiental com $x = 25$	Fonte	165.37	99,3
Oclusão Ambiental com $x = 36$	Fonte	231.86	99,5
Oclusão Ambiental com $x = 49$	Fonte	309.13	99,6
Oclusão Ambiental com $x = 64$	Fonte	397.88	99,7
Oclusão Ambiental com $x = 81$	Fonte	497.68	99,7
Oclusão Ambiental com $x = 100$	Fonte	607.87	99,8
Todos excepto Oclusão Ambiental	Fonte	17.93	94,5
Todos	Fonte	759.74	99,8

Tabela 2: Efeitos e respetivo desempenho.

Como é possível ver pela Tabela 2, o custo por simplesmente interligar *OpenGL* com *Optix* só por si já é muito elevado. Isto só por si é algo que desencoraja a sua utilização.

Os custos associados a utilizar cada um dos efeitos também são algo elevados, sendo que a oclusão ambiental é o efeito mais prejudicial ao desempenho, porém confere muito mais realismo e profundidade à cena. As sombras, reflexões e refrações são relativamente baratas e não põe em causa a geração de imagens em tempo real para esta cena usada nos testes. Como era de esperar, o peso absoluto da rasterização mantém-se constante em todas as cenas, pelo que o peso relativo do *Optix* é o mais significativo.

É também possível verificar que o tempo de desenhar uma frame com todos os efeitos é superior do que a soma dos tempos de cada um dos efeitos individualmente. Isto acontece porque para cada raio refletido e refratado é necessário calcular a se o ponto atingido está em sombra e qual o seu nível de exposição.

Futuros trabalhos envolveriam a comparação de desempenho destes mesmos efeitos implementados com rasterização, bem como das diferenças nas imagens produzidas, e tentar estabelecer um equilíbrio entre que efeitos poderiam ser implementados sob rasterização ou *Ray-Tracing*.

Referências

- Nvidia. Nvidia optix ray tracing engine, November 2012. URL <https://developer.nvidia.com/optix>.
- António Ramires. Cube mapping and glsl, a.
- António Ramires. Lighthouse3d, b. URL <http://www.lighthouse3d.com/>.

A Ficheiros XML

A.1 SceneNoEffects.xml

```
<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file ..\3dsmodels\cow
                    .obj-->
                <!--file ..\ntg-bench\
                    bench.obj-->
                <file ..\3dsmodels\
                    largoCamoess.cbo</file
                >
                <!--file ..\3dsmodels\
                    pl3dsecXXI-octreeByMat
                    -test.cbo-->
                <!--file ..\ntg-bin-pl3d-
                    secxxi-other\
                    avPlatanosArvoresII.
                    mesh.xml-->
                <!--file ..\3dsmodels\
                    jeep1.3ds-->
            </scene>
        </scenes>
        <viewports>
            <viewport name="Viewport" fixed="true">
                <geometry x="0" y="0"
                    width="1.0" height
                    ="1.0" />
                <bcolor r="0.0" g="0.0"
                    b="0.0" />
            </viewport>
        </viewports>
        <cameras>
            <camera name="MainCamera" type="perspective">
                <viewport>Viewport</
                    viewport>
                <perspective fov="60.0"
                    near="0.3" far
                    ="1000.0" />
                <!--position x="0.0" y
                    ="0.0" z="15.0" /-->
                <position x="7.42" y
                    ="18.61" z="-38.62" />
                <view x="0.31" y=" -0.36"
```

```

        z="0.87" />
    <up x="0.0" y="1.0" z
        ="0.0" />
</camera>
</cameras>
<lights>
    <light name="Sun" type="
        directional">
        <position x="-52" y
            ="77.0" z="-27.0" />
        <direction x="0.597" y
            ="-0.390" z="0.700" />
        <color r="0.9" g="0.9" b
            ="0.9" />
        <ambient r="0.5" g="0.5"
            b="0.5" />
    </light>
</lights>
<materialLibs>
    <mlib filename=".\\mlibs\\
        optixTest.mlib"/>
    <mlib filename=".\\mlibs\\
        DeferredRenderTarget.mlib"/>
</materialLibs>
</assets>

```

```

<pipelines>

    <pipeline name="fixedfunction" default="
        true" defaultCamera="MainCamera">
        <pass class="default" name="pass1
            ">
            <viewport>Viewport</
                viewport>
            <scenes>
                <scene>MainScene
                    </scene>
            </scenes>
            <camera>MainCamera</
                camera>

            <lights>
                <light>Sun</light
                    >
            </lights>
            <injectionMaps>
                <map toMaterial

```

```

=*>
<shader
  fromMaterial
  =”
  deferred
  ”
  fromLibrary
  =”
  Deferred

  Render

  Targets
  ” />
    </map>
  </injectionMaps>
</pass>

</pipeline>
</pipelines>
</project>

```

A.2 SceneDummy.xml

```

<?xml version=”1.0” ?>
<project name=”teste1” width=1024 height=1024>
  <assets>
    <scenes>
      <scene name=”MainScene”>
        <!--file >..\3dsmodels\cow
          .obj</file -->
        <!--file >..\ntg-bench\
          bench.obj</file -->
        <file >..\3dsmodels\
          largoCamoos.cbo</file >
        <!--file >..\3dsmodels\
          pl3dsecXXI-octreeByMat
          -test.cbo</file -->
        <!--file >..\ntg-bin-pl3d-
          secxxi-other\
          avPlatanosArvoresII.
          mesh.xml</file -->
        <!--file >..\3dsmodels\
          jeep1.3ds</file -->
      </scene>
    </scenes>
    <viewports>
      <viewport name=”Viewport” fixed=”
        true”>
        <geometry x=”0” y=”0”
```

```

        width="1.0" height
        ="1.0" />
    <bgcolor r="0.0" g="0.0"
        b="0.0" />
</viewport>
</viewports>
<cameras>
    <camera name="MainCamera" type="

perspective">
        <viewport>Viewport</
            viewport>
        <perspective fov="60.0"
            near="0.3" far
            ="1000.0" />
        <!--position x="0.0" y
            ="0.0" z="15.0" -->
        <position x="7.42" y
            ="18.61" z="-38.62" />
        <view x="0.31" y=""-0.36"
            z="0.87" />
        <up x="0.0" y="1.0" z
            ="0.0" />
    </camera>
</cameras>
<lights>
    <light name="Sun" type="

directional">
        <position x="-52" y
            ="77.0" z=""-27.0" />
        <direction x="0.597" y
            ="-0.390" z="0.700" />
        <color r="0.9" g="0.9" b
            ="0.9" />
        <ambient r="0.5" g="0.5"
            b="0.5" />
    </light>
</lights>
<materialLibs>
    <mllib filename="..\mlibs\

optixTest.mlib"/>
    <mllib filename="..\mlibs\

DeferredRenderTarget.mlib"/>
</materialLibs>
</assets>

<pipelines>
```

```

<pipeline name="fixedfunction" default="true" defaultCamera="MainCamera">
    <pass class="default" name="pass1">
        <scenes>
            <scene>MainScene
            </scene>
        </scenes>
        <camera>MainCamera</camera>
        <rendertarget name="deferred" fromLibrary="Deferred Render Targets" />
        <lights>
            <light>Sun</light>
        </lights>
        <injectionMaps>
            <map toMaterial="*>
                <shader
                    fromMaterial=""
                    fromLibrary=""
                    ="" deferred=""
                    ="" Deferred=""
                    ="" Targets=""
                    ="" />
            </map>
        </injectionMaps>
    </pass>

    <pass class="optix" name="pass2">
        <scenes>
            <scene>MainScene
            </scene>
        </scenes>
        <camera>MainCamera</camera>
        <rendertarget name="test" fromLibrary="Optix Ray Tracer Render Target" />
    </pass>

```

```

<lights>
    <light>Sun</light>
</lights>
<materialMaps>
    <!--map
        fromMaterial = "*" toLibrary =
        "curitiba_material_lib"
        " toMaterial=" ..Black" /-->
    </materialMaps>

<optixEntryPoint>
    <optixProgram
        type="RayGen"
        file="optix/dummy.ptx"
        proc="buffer_camera"/>
    <optixProgram
        type="Exception"
        file="optix/dummy.ptx"
        proc="exception"/>
</optixEntryPoint>

<!--optixDefaultMaterial>
    <optixProgram
        type="Closest_Hit"
        ray="Phong" file="optix/dummy.ptx"
        proc="closest_hit"/>
    <optixProgram
        type="Miss"
        ray="Phong" file="optix/dummy.ptx"
        proc="miss"/>
    <optixProgram
        type="Any_Hit">

```

```

        ray="Shadow"
        file="optix/
dummy.ptx"
proc="
any_hit_shadow
"/>
<optixProgram
type="Miss"
ray="Shadow" file="optix/dummy.
ptx" proc="
miss_shadow"/>
</optixDefaultMaterial-->

<optixGeometryProgram>
    <optixProgram
        type="Geometry_Intersection
" file="optix/dummy.
ptx" proc="
geometryintersection
"/>
    <optixProgram
        type="Bounding_Box"
        file="optix/
dummy.ptx"
        proc="
boundingbox"/>
</optixGeometryProgram>
<optixVertexAttributes>
    <attribute name="position"/>
    <attribute name="normal"/>
    <attribute name="texCoord0"/>
</optixVertexAttributes>

<optixMaterialAttributes>
    <valueof optixVar
        ="diffuse"
        type="CURRENT"
        context="
COLOR"
        component="
DIFFUSE" />
    <valueof optixVar

```

```

        ="texCount"
        type="CURRENT"
        context=""
        TEXTURE"
        component=""
        COUNT" />
    </optixMaterialAttributes
    >

    <optixGlobalAttributes>
        <valueof optixVar
            ="lightDir"
            type="CURRENT"
            context=""
            LIGHT" id=0
            component=""
            DIRECTION" />
        <valueof optixVar
            ="lightColor"
            type="CURRENT"
            context=""
            LIGHT" id=0
            component=""
            DIFFUSE" />
    </optixGlobalAttributes>

    <optixInput>
        <buffer var=""
            pos_buffer"
            texture=""
            Deferred
            Render Targets
            :: pos" />
        <buffer var=""
            norm_buffer"
            texture=""
            Deferred
            Render Targets
            :: norm" />
        <buffer var=""
            color_buffer"
            texture=""
            Deferred
            Render Targets
            :: color" />
    </optixInput>
</pass>

<pass class="quad" name="pass5">
    <viewport>Viewport</

```

```

        <viewport>
    <texture name="offscreenrender"
        fromLibrary="Optix Ray
        Tracer Render Target"
    />
    <!--materialMaps>
        <map fromMaterial
            ="_Quad"
            toLibrary="Deferred
            Render Targets"
            toMaterial="combine" />
    </materialMaps-->
    </pass>
</pipeline>
</pipelines>
</project>

```

A.3 SceneShadows.xml

```

<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file >..\3dsmodels\cow
                    .obj</file -->
                <!--file >..\ntg-bench\
                    bench.obj</file -->
                <file >..\3dsmodels\
                    largoCamoess.cbo</file
                >
                <!--file >..\3dsmodels\
                    pl3dsecXXI-octreeByMat
                    -test.cbo</file -->
                <!--file >..\ntg-bin-pl3d-
                    secxxi-other\
                    avPlatanosArvoresII.
                    mesh.xml</file -->
                <!--file >..\3dsmodels\
                    jeep1.3ds</file -->
            </scene>
        </scenes>
        <viewports>
            <viewport name="Viewport" fixed="
                true">
                <geometry x="0" y="0"
                    width="1.0" height

```

```

                =”1.0” />
            <bgcolor r=”0.0” g=”0.5”
                    b=”0.0” />
        </viewport>
    </viewports>
<cameras>
    <camera name=”MainCamera” type=”
            perspective”>
        <viewport>Viewport</
            viewport>
        <perspective fov=”60.0”
            near=”0.3” far
            =”1000.0” />
        <!--position x=”0.0” y
            =”0.0” z=”15.0” -->
        <position x=”7.42” y
            =”18.61” z=”-38.62” />
        <view x=”0.31” y=”-0.36”
            z=”0.87” />
        <up x=”0.0” y=”1.0” z
            =”0.0” />
    </camera>
</cameras>
<lights>
    <light name=”Sun” type=”
            directional”>
        <position x=”-52” y
            =”77.0” z=”-27.0” />
        <direction x=”0.597” y
            =”-0.390” z=”0.700” />
        <color r=”0.9” g=”0.9” b
            =”0.9” />
        <ambient r=”0.5” g=”0.5”
            b=”0.5” />
    </light>
</lights>
<materialLibs>
    <mllib filename=”..\\mlibs\\
            optixTest.mlib”/>
    <mllib filename=”..\\mlibs\\
            DeferredRenderTarget.mlib”/>
</materialLibs>
</assets>

<pipelines>
    <pipeline name=”fixedfunction” default=”
```

```

        true" defaultCamera="MainCamera">
        <pass class="default" name="pass1">
            <scenes>
                <scene>MainScene
                </scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="deferred" fromLibrary="Deferred Render Targets" />
            <lights>
                <light>Sun</light>
            </lights>
            <injectionMaps>
                <map toMaterial="*>
                    <shader
                        fromMaterial=""
                        ="
                        deferred"
                        fromLibrary=""
                        ="
                        Deferred
                        Render
                        Targets
                        " />
                </map>
            </injectionMaps>
        </pass>

        <pass class="optix" name="pass2">
            <scenes>
                <scene>MainScene
                </scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="test" fromLibrary="Optix Ray Tracer Render Target" />
            <lights>

```

```

<light>Sun</light>
</lights>
<materialMaps>
    <map fromMaterial="*" toLibrary="curitiba_material_lib" toMaterial="__Black" />
</materialMaps>
<optixEntryPoint>
    <optixProgram type="RayGen" file="optix/deferredShadows.ptx" proc="buffer_camera"/>
    <optixProgram type="Exception" file="optix/deferredShadows.ptx" proc="exception"/>
</optixEntryPoint>
<optixDefaultMaterial>
    <optixProgram type="Any_Hit" ray="Shadow" file="optix/deferredShadows.ptx" proc="any_hit_shadow"/>
</optixDefaultMaterial>
<optixGeometryProgram>
    <optixProgram type="Geometry_Intersection" file="optix/deferredShadows.ptx" proc="geometryintersection"/>
    <optixProgram type=""

```

```

        Bounding_Box"
            file="optix/
            deferredShadows
            .ptx" proc="
            boundingbox"/>
    </optixGeometryProgram>
    <optixVertexAttributes>
        <attribute name="
            position"/>
    </optixVertexAttributes>
    <optixGlobalAttributes>
        <valueof optixVar
            ="lightDir"
            type="CURRENT"
            context="
            LIGHT" id=0
            component="
            DIRECTION" />
    </optixGlobalAttributes>
    <optixInput>
        <buffer var="
            pos_buffer"
            texture="
            Deferred
            Render Targets
            :: pos" />
    </optixInput>
</pass>

<pass class="quad" name="pass5">
    <viewport>Viewport</
        viewport>
    <!--texture name="
        offscreenrender"
        fromLibrary="Optix Ray
        Tracer Render Target"
        /-->
    <materialMaps>
        <map fromMaterial
            ="_Quad"
            toLibrary="
            Deferred
            Render Targets
            " toMaterial="
            combine" />
    </materialMaps>
    </pass>
</pipeline>
</pipelines>
</project>

```

A.4 SceneReflect.xml

```
<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file >..\3dsmodels\cow
                    .obj</file -->
                <!--file >..\ntg-bench\
                    bench.obj</file -->
                <file >..\3dsmodels\
                    largoCamoess.cbo</file >
                <!--file >..\3dsmodels\
                    pl3dsecXXI-octreeByMat
                    -test.cbo</file -->
                <!--file >..\ntg-bin-pl3d-
                    secxxi-other\
                    avPlatanosArvoresII.
                    mesh.xml</file -->
                <!--file >..\3dsmodels\
                    jeep1.3ds</file -->
            </scene>
        </scenes>
        <viewports>
            <viewport name="Viewport" fixed="true">
                <geometry x="0" y="0"
                    width="1.0" height
                    ="1.0" />
                <bgcolor r="0.0" g="0.5"
                    b="0.0" />
            </viewport>
        </viewports>
        <cameras>
            <camera name="MainCamera" type="perspective">
                <viewport>Viewport</
                    viewport>
                <perspective fov="60.0"
                    near="0.3" far
                    ="1000.0" />
                <!--position x="0.0" y
                    ="0.0" z="15.0" -->
                <position x="7.42" y
                    ="18.61" z="-38.62" />
                <view x="0.31" y="-0.36"
                    z="0.87" />
                <up x="0.0" y="1.0" z
                    ="0.0" />
            </camera>
        </cameras>
    </assets>
</project>
```

```

        </camera>
    </cameras>
    <lights>
        <light name="Sun" type="directional">
            <position x="-52" y="77.0" z="-27.0" />
            <direction x="0.597" y="-0.390" z="0.700" />
            <color r="0.9" g="0.9" b="0.9" />
            <ambient r="0.5" g="0.5" b="0.5" />
        </light>
    </lights>
    <materialLibs>
        <mlib filename=".\\mlibs\\optixTest.mlib"/>
        <mlib filename=".\\mlibs\\DeferredRenderTargetTargets.mlib"/>
    </materialLibs>
</assets>

<pipelines>
    <pipeline name="fixedfunction" default="true" defaultCamera="MainCamera">
        <pass class="default" name="pass1">
            <scenes>
                <scene>MainScene
                </scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="deferred" fromLibrary="Deferred Render Targets" />
            <lights>
                <light>Sun</light>
            </lights>
            <injectionMaps>
                <map toMaterial="*"/>
            <shader>

```

```

        fromMaterial
        =”
        deferred
        ”
        fromLibrary
        =”
        Deferred

        Render

        Targets
        ” />
</map>
<map toMaterial=”
      Vidro”>
<color
      fromMaterial
      =”
      deferredVidro
      ”
      fromLibrary
      =”
      Deferred

      Render

      Targets
      ”
      ambient
      =true
      diffuse
      =true
      emission
      =true
      specular
      =true
      shininess
      =false
      />
</map>
</injectionMaps>
</pass>

<pass class=”optix” name=”pass2”>
<scenes>
  <scene>MainScene
  </scene>
</scenes>
<camera>MainCamera</

```

```

        camera>
<rendertarget name="test"
    fromLibrary="Optix
    Ray Tracer Render
    Target" />
<lights>
    <light>Sun</light
    >
</lights>
<materialMaps>
    <!--map
        fromMaterial
        = "*" toLibrary
        =
        curitiba_material.lib
        " toMaterial="
        _Black" /-->
</materialMaps>

<optixEntryPoint>
    <optixProgram
        type="RayGen"
        file="optix/
        reflection.ptx
        " proc="
        buffer_camera
        "/>
    <optixProgram
        type="
        Exception"
        file="optix/
        reflection.ptx
        " proc="
        exception"/>
</optixEntryPoint>

<optixDefaultMaterial>
    <optixProgram
        type="
        Closest_Hit"
        ray="
        Phong" file="
        optix/
        reflection.ptx
        " proc="
        closest_hit"/>
    <optixProgram
        type="Miss"
        ray="

```

```

        Phong" file=""
        optix/
        reflection.ptx
        " proc="miss
        "/>
</optixDefaultMaterial>

<optixMaterialMap>
    <optixMap to="curitiba_material_lib:Vidro">
        <
            optixProgram
            type =
            =
            Closest_Hit
            ,

            ray="Phong"
            file
            =
            optix/
            reflection
            .ptx"
            proc=
            closest_hit_glass
            "/>
        </optixMap>
    </optixMaterialMap>

    <optixGeometryProgram>
        <optixProgram
            type="Geometry_Intersection"
            " file="optix/
            reflection.ptx
            " proc="geometryintersection
            "/>
        <optixProgram
            type="Bounding_Box"
            file="optix/
            reflection.ptx
            " proc="boundingbox"/>
    </optixGeometryProgram>
    <optixVertexAttributes>

```

```

<attribute name="position"/>
<attribute name="normal"/>
<attribute name="texCoord0"/>
</optixVertexAttributes>

<optixMaterialAttributes>
    <valueof optixVar="diffuse"
        type="CURRENT"
        context="COLOR"
        component="DIFFUSE" />
    <valueof optixVar="texCount"
        type="CURRENT"
        context="TEXTURE"
        component="COUNT" />
</optixMaterialAttributes
>

<optixGlobalAttributes>
    <valueof optixVar="lightDir"
        type="CURRENT"
        context="LIGHT" id=0
        component="DIRECTION" />
    <valueof optixVar="lightColor"
        type="CURRENT"
        context="LIGHT" id=0
        component="DIFFUSE" />
</optixGlobalAttributes>

<optixInput>
    <buffer var="pos_buffer"
        texture="Deferred
        Render Targets
        :: pos" />

```

```

<buffer var="norm_buffer"
        texture="Deferred
Render Targets
::norm" />
<buffer var="color_buffer"
        texture="Deferred
Render Targets
::color" />
</optixInput>
</pass>

<pass class="quad" name="pass5">
    <viewport>Viewport</viewport>
    <texture name="offscreenrender"
            fromLibrary="Optix Ray
Tracer Render Target"
            />
    <!--materialMaps>
        <map fromMaterial
            ="__Quad"
            toLibrary="Deferred
Render Targets
" toMaterial="combine" />
    </materialMaps-->
    </pass>
</pipeline>
</pipelines>
</project>

```

A.5 SceneRefract.xml

```

<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file >..\3dsmodels\cow
                    .obj</file-->
                <!--file >..\ntg-bench\
                    bench.obj</file-->
                <file >..\3dsmodels\
                    largoCamoes.cbo</file>

```

```

<!--file >..\3dsmodels\
pl3dsecXXI-octreeByMat
-test.cbo</file -->
<!--file >..\ntg-bin-pl3d-
secxxi-other\
avPlatanosArvoresII.
mesh.xml</file -->
<!--file >..\3dsmodels\
jeep1.3ds</file -->

```

 </scene>

```

</scenes>
<viewports>
  <viewport name="Viewport" fixed="true">
    <geometry x="0" y="0"
              width="1.0" height
              ="1.0" />
    <bgcolor r="0.0" g="0.5"
             b="0.0" />
  </viewport>
</viewports>
<cameras>
  <camera name="MainCamera" type="perspective">
    <viewport>Viewport</viewport>
    <perspective fov="60.0"
                  near="0.3" far
                  ="1000.0" />
    <!--position x="0.0" y
                  ="0.0" z="15.0" /-->
    <position x="7.42" y
              ="18.61" z="-38.62" />
    <view x="0.31" y=" -0.36"
          z="0.87" />
    <up x="0.0" y="1.0" z
        ="0.0" />
  </camera>
</cameras>
<lights>
  <light name="Sun" type="directional">
    <position x="-52" y
              ="77.0" z="-27.0" />
    <direction x="0.597" y
               ="-0.390" z="0.700" />
    <color r="0.9" g="0.9" b
           ="0.9" />
    <ambient r="0.5" g="0.5"
              b="0.5" />
  </light>
</lights>

```

```

        </light>
    </lights>
    <materialLibs>
        <mlib filename="..\mlibs\optixTest.mlib"/>
        <mlib filename="..\mlibs\DeferredRenderTarget.mlib"/>
    </materialLibs>
</assets>

<pipelines>
    <pipeline name="fixedfunction" default="true" defaultCamera="MainCamera">
        <pass class="default" name="pass1">
            <scenes>
                <scene>MainScene</scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="deferred" fromLibrary="Deferred Render Targets" />
            <lights>
                <light>Sun</light>
            </lights>
            <injectionMaps>
                <map toMaterial="*">
                    <shader fromMaterial="deferred" fromLibrary="Deferred Render Targets" />
                </map>
            </injectionMaps>
        </pass>
    </pipeline>
</pipelines>

```

```

<map toMaterial="Vidro">
    <color
        fromMaterial =
        ="deferredVidro"
        "fromLibrary =
        ="Deferred
        Render
        Targets
        "
        ambient
        =true
        diffuse
        =true
        emission
        =true
        specular
        =true
        shininess
        =false
        />
    </map>
</injectionMaps>
</pass>

<pass class="optix" name="pass2">
    <scenes>
        <scene>MainScene
        </scene>
    </scenes>
    <camera>MainCamera</
        camera>
    <rendertarget name="test"
        fromLibrary="Optix
        Ray Tracer Render
        Target" />
    <lights>
        <light>Sun</light
        >
    </lights>
    <materialMaps>
        <!--map
            fromMaterial
            =""*"" toLibrary

```

```

        ="curitiba_material_lib"
        " toMaterial="
        ..Black" /-->
    </materialMaps>

<optixEntryPoint>
    <optixProgram
        type="RayGen"
        file="optix/
refraction.ptx"
        " proc="
        buffer_camera
    "/>
    <optixProgram
        type="Exception"
        file="optix/
refraction.ptx"
        " proc="
        exception"/>
</optixEntryPoint>

<optixDefaultMaterial>
    <optixProgram
        type="Closest_Hit"
        ray="Phong" file="optix/
refraction.ptx"
        " proc="
        closest_hit"/>
    <optixProgram
        type="Miss"
        ray="Phong" file="optix/
refraction.ptx"
        " proc="miss
    "/>
</optixDefaultMaterial>

<optixMaterialMap>
    <optixMap to="
        curitiba_material_lib
        : Vidro">
        <
            optixProgram

```

```

        type
      ="
      Closest_Hit
      "

      ray="
      Phong"
      file
      ="optix/
      refraction
      .ptx"
      proc="
      closest_hit_glass
      "/>
    </optixMap>
  </optixMaterialMap>

<optixGeometryProgram>
  <optixProgram
    type="
    Geometry_Intersection
    " file="
    optix/
    refraction.ptx
    " proc="
    geometryintersection
    "/>
  <optixProgram
    type="
    Bounding_Box"
    file="optix/
    refraction.ptx
    " proc="
    boundingbox"/>
</optixGeometryProgram>
<optixVertexAttributes>
  <attribute name="
    position"/>
  <attribute name="
    normal"/>
  <attribute name="
    texCoord0"/>
</optixVertexAttributes>

<optixMaterialAttributes>
  <valueof optixVar
  ="diffuse"
  type="CURRENT"
  context="
```

```

        COLOR"
        component=""
        DIFFUSE" />
    <valueof optixVar
        ="texCount"
        type="CURRENT"
        context=""
        TEXTURE"
        component=""
        COUNT" />
</optixMaterialAttributes
>

<optixGlobalAttributes>
    <valueof optixVar
        ="lightDir"
        type="CURRENT"
        context=""
        LIGHT" id=0
        component=""
        DIRECTION" />
    <valueof optixVar
        ="lightColor"
        type="CURRENT"
        context=""
        LIGHT" id=0
        component=""
        DIFFUSE" />
</optixGlobalAttributes>

<optixInput>
    <buffer var=""
        pos_buffer"
        texture=""
        Deferred
        Render Targets
        :: pos" />
    <buffer var=""
        norm_buffer"
        texture=""
        Deferred
        Render Targets
        :: norm" />
    <buffer var=""
        color_buffer"
        texture=""
        Deferred
        Render Targets
        :: color" />
</optixInput>

```

```

        </pass>

        <pass class="quad" name="pass5">
            <viewport>Viewport</
                viewport>
            <texture name="
                offscreenrender"
                fromLibrary="Optix Ray
                Tracer Render Target"
                />
        <!--materialMaps>
            <map fromMaterial
                ="_Quad"
                toLibrary="
                    Deferred
                    Render Targets
                " toMaterial="
                    combine" />
        </materialMaps-->
    </pass>
</pipeline>
</pipelines>
</project>

```

A.6 SceneSSAO.xml

```

<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file >..\3dsmodels\cow
                    .obj</file -->
                <!--file >..\ntg-bench\
                    bench.obj</file -->
                <file >..\3dsmodels\
                    largoCamoos.cbo</file
                >—
                <!--file >..\3dsmodels\
                    pl3dsecXXI-octreeByMat
                    -test.cbo</file -->
                <!--file >..\ntg-bin-pl3d-
                    secxxi-other\
                    avPlatanosArvoresII.
                    mesh.xml</file -->
                <!--file >..\3dsmodels\
                    jeep1.3ds</file -->
            </scene>
        </scenes>
        <viewports>

```

```

<viewport name="Viewport" fixed="true">
    <geometry x="0" y="0"
              width="1.0" height
              ="1.0" />
    <bgcolor r="0.0" g="0.5"
              b="0.0" />
</viewport>
</viewports>
<cameras>
    <camera name="MainCamera" type="perspective">
        <viewport>Viewport</viewport>
        <perspective fov="60.0"
                      near="0.3" far
                      ="1000.0" />
        <!--position x="0.0" y
                      ="0.0" z="15.0" /-->
        <position x="7.42" y
                  ="18.61" z=" -38.62" />
        <view x="0.31" y=" -0.36"
              z="0.87" />
        <up x="0.0" y="1.0" z
            ="0.0" />
    </camera>
</cameras>
<lights>
    <light name="Sun" type="directional">
        <position x="-52" y
                  ="77.0" z=" -27.0" />
        <direction x="0.597" y
                  =" -0.390" z="0.700" />
        <color r="0.9" g="0.9" b
               ="0.9" />
        <ambient r="0.5" g="0.5"
                  b="0.5" />
    </light>
</lights>
<materialLibs>
    <mlib filename="..\mlibs\optixTest.mlib"/>
    <mlib filename="..\mlibs\DeferredRenderTarget.mlib"/>
</materialLibs>
</assets>

```

```

<pipelines>

    <pipeline name="fixedfunction" default="true" defaultCamera="MainCamera">
        <pass class="default" name="pass1">
            <scenes>
                <scene>MainScene</scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="deferred" fromLibrary="Deferred Render Targets" />
            <lights>
                <light>Sun</light>
            </lights>
            <injectionMaps>
                <map toMaterial="*">
                    <shader fromMaterial="deferred" fromLibrary="Deferred" />
                    <Targets />
                </map>
            </injectionMaps>
        </pass>

        <pass class="optix" name="pass2">
            <scenes>
                <scene>MainScene</scene>
            </scenes>
            <camera>MainCamera</camera>
            <rendertarget name="test" />
        </pass>
    </pipeline>

```

```

        fromLibrary="Optix
Ray Tracer Render
Target" />
<materialMaps>
    <map fromMaterial
        = "*" toLibrary
        =
        curitiba_material_lib
        " toMaterial="
        ..Black" />
</materialMaps>
<optixEntryPoint>
    <optixProgram
        type="RayGen"
        file="optix/
        ssao.ptx" proc
        =
        buffer_camera
        "/>
    <optixProgram
        type=
        Exception"
        file="optix/
        ssao.ptx" proc
        ="exception"/>
</optixEntryPoint>
<optixDefaultMaterial>
    <optixProgram
        type="Any_Hit"
        ray="
        Shadow" file="
        optix/ssao.ptx
        " proc="
        any_hit_shadow
        "/>
    <optixProgram
        type="Miss"
        ray="
        Shadow" file="
        optix/ssao.ptx
        " proc="miss
        "/>
</optixDefaultMaterial>
<optixGeometryProgram>
    <optixProgram
        type="
        Geometry_Intersection
        " file="
        optix/ssao.ptx

```

```

        " proc="
        geometryintersection
    "/>
<optixProgram
    type="
    Bounding_Box"
    file="optix/
    ssao.ptx" proc
    ="boundingbox
    "/>
</optixGeometryProgram>
<optixVertexAttributes>
    <attribute name="
    position"/>
</optixVertexAttributes>
<optixInput>
    <buffer var="
    pos_buffer"
    texture="
    Deferred
    Render Targets
    :: pos" />
    <buffer var="
    norm_buffer"
    texture="
    Deferred
    Render Targets
    :: norm" />
</optixInput>
</pass>

<pass class="quad" name="pass5">
    <viewport>Viewport</
        viewport>
    <!--texture name="
    offscreenrender"
    fromLibrary="Optix Ray
    Tracer Render Target"
    /-->
    <materialMaps>
        <map fromMaterial
            ="_Quad"
            toLibrary="
            Deferred
            Render Targets"
            toMaterial="
            combine" />
    </materialMaps>
</pass>
</pipeline>

```

```

        </pipelines>
</project>
```

A.7 SceneFinal.xml

```

<?xml version="1.0" ?>
<project name="teste1" width=1024 height=1024>
    <assets>
        <scenes>
            <scene name="MainScene">
                <!--file >..\3dsmodels\cow
                    .obj</file -->
                <!--file >..\ntg-bench\
                    bench.obj</file -->
                <file >..\3dsmodels\
                    largoCamoess.cbo</file >
                <!--file >..\3dsmodels\
                    pl3dsecXXI-octreeByMat
                    -test.cbo</file -->
                <!--file >..\ntg-bin-pl3d-
                    secxxi-other\
                    avPlatanosArvoresII.
                    mesh.xml</file -->
                <!--file >..\3dsmodels\
                    jeep1.3ds</file -->
            </scene>
        </scenes>
        <viewports>
            <viewport name="Viewport" fixed="true">
                <geometry x="0" y="0"
                    width="1.0" height
                    ="1.0" />
                <bgcolor r="0.0" g="0.0"
                    b="0.0" />
            </viewport>
        </viewports>
        <cameras>
            <camera name="MainCamera" type="perspective">
                <viewport>Viewport</
                    viewport>
                <perspective fov="60.0"
                    near="0.3" far
                    ="1000.0" />
                <!--position x="0.0" y
                    ="0.0" z="15.0" -->
                <position x="7.42" y
                    ="18.61" z="-38.62" />
                <view x="0.31" y="
```

```

        z="0.87" />
    <up x="0.0" y="1.0" z
        ="0.0" />
</camera>
</cameras>
<lights>
    <light name="Sun" type="
        directional">
        <position x="-52" y
            ="77.0" z="-27.0" />
        <direction x="0.597" y
            ="-0.390" z="0.700" />
        <color r="0.9" g="0.9" b
            ="0.9" />
        <ambient r="0.5" g="0.5"
            b="0.5" />
    </light>
</lights>
<materialLibs>
    <mlib filename=".\\mlibs\\
        optixTest.mlib"/>
    <mlib filename=".\\mlibs\\
        DeferredRenderTarget.mlib"/>
</materialLibs>
</assets>

```

```

<pipelines>

    <pipeline name="fixedfunction" default="
        true" defaultCamera="MainCamera">
        <pass class="default" name="pass1
            ">
            <scenes>
                <scene>MainScene
                </scene>
            </scenes>
            <camera>MainCamera</
                camera>
            <rendertarget name="
                deferred" fromLibrary
                ="Deferred Render
                Targets" />
            <lights>
                <light>Sun</light
                >
            </lights>
            <injectionMaps>

```

```

<map toMaterial
      = "*" >
    <shader
      fromMaterial
      = ""
      deferred
      =
      fromLibrary
      = ""
      Deferred

      Render

      Targets
      " />
    </map>
    <map toMaterial="Vidro">
      <color
        fromMaterial
        = ""
        deferredVidro
        =
        fromLibrary
        = ""
        Deferred

        Render

        Targets
        "
        ambient
        = true
        diffuse
        = true
        emission
        = true
        specular
        = true
        shininess
        = false
        />
      </map>
    </injectionMaps>
  </pass>

  <pass class="optix" name="pass2">
    <scenes>
      <scene>MainScene

```

```

        </scene>
    </scenes>
    <camera>MainCamera</
        camera>
    <rendertarget name="test"
        fromLibrary="Optix
        Ray Tracer Render
        Target" />
    <lights>
        <light>Sun</light
        >
    </lights>
    <materialMaps>
        <!--map
            fromMaterial
            = "*" toLibrary
            =
            curitiba_material_lib
            " toMaterial="
            ..Black" /-->
    </materialMaps>

    <optixEntryPoint>
        <optixProgram
            type="RayGen"
            file="optix/
            final.ptx"
            proc="
            buffer_camera
            "/>
        <optixProgram
            type="
            Exception"
            file="optix/
            final.ptx"
            proc="
            exception"/>
    </optixEntryPoint>

    <optixDefaultMaterial>
        <optixProgram
            type="
            Closest_Hit"
            ray="
            Phong" file="
            optix/final.
            ptx" proc="
            closest_hit"/>
    <optixProgram

```

```

        type="Miss"
        ray="
Phong" file="
optix/final.
ptx" proc="
miss"/>
<optixProgram
        type="Any_Hit"
        ray="Shadow"
        file="optix/
final.ptx"
        proc="
any_hit_shadow
"/>
<optixProgram
        type="Miss"
        ray="
Shadow" file="
optix/final.
ptx" proc="
miss_shadow"/>
</optixDefaultMaterial>

<optixMaterialMap>
    <optixMap to="
curitiba_material_lib
:Vidro">
        <
            optixProgram
            type
            =""
            Closest_Hit
            "
            ray="
Phong"
            file
            =""
            optix/
            final.
            ptx"
            proc="
closest_hit_glass
"/>
        <
            optixProgram
            type
            =""
            Any_Hit
            " ray

```

```

        =”
        Shadow
        ” file
        =”
        optix/
        final.
        ptx”
        proc=”
        any_hit_shadow_glass
        ”/”

    </optixMap>
</optixMaterialMap>

<optixGeometryProgram>
    <optixProgram
        type=”
        Geometry_Intersection
        ” file=”
        optix/final.
        ptx” proc=”
        geometryintersection
        ”/”>
    <optixProgram
        type=”
        Bounding_Box”
        file=”optix/
        final.ptx”
        proc=”
        boundingbox”/>
</optixGeometryProgram>
<optixVertexAttributes>
    <attribute name=”
        position”/>
    <attribute name=”
        normal”/>
    <attribute name=”
        texCoord0”/>
</optixVertexAttributes>

<optixMaterialAttributes>
    <valueof optixVar
        =”diffuse”
        type=”CURRENT”
        context=”
        COLOR”
        component=”
        DIFFUSE” />
    <valueof optixVar
        =”texCount”
        type=”CURRENT”
```

```

        context="TEXTURE"
        component="COUNT" />
    </optixMaterialAttributes
    >

<optixGlobalAttributes>
    <valueof optixVar
        ="lightDir"
        type="CURRENT"
        context="LIGHT" id=0
        component="DIRECTION" />
    <valueof optixVar
        ="lightColor"
        type="CURRENT"
        context="LIGHT" id=0
        component="DIFFUSE" />
</optixGlobalAttributes>

<optixInput>
    <buffer var="pos_buffer"
        texture="Deferred
        Render Targets
        :: pos" />
    <buffer var="norm_buffer"
        texture="Deferred
        Render Targets
        :: norm" />
    <buffer var="color_buffer"
        texture="Deferred
        Render Targets
        :: color" />
</optixInput>
</pass>

<pass class="quad" name="pass5">
    <viewport>Viewport</
        viewport>
    <texture name="

```

```

        offscreenrender"
        fromLibrary="Optix Ray
                    Tracer Render Target"
        />
<!--materialMaps>
    <map fromMaterial
        ="__Quad"
        toLibrary="
        Deferred
        Render Targets
        " toMaterial="
        combine" />
</materialMaps-->
</pass>
</pipeline>
</pipelines>
</project>

```

B Ficheiros Cuda

B.1 deferredShadows.cu

```

#include <optix.h>
#include <optixu/optixu_math_namespace.h>
#include <optixu/optixu_matrix_namespace.h>
#include <optixu/optixu_aabb.h>
#include <optixu/optixu_aabb_namespace.h>

using namespace optix;

struct PerRayDataResult
{
    float4 result;
};

rtDeclareVariable(float4, lightDir, , );
rtDeclareVariable(rtObject, top_object, , );
rtBuffer<float4> vertex_buffer;
rtBuffer<uint> index_buffer;

rtTextureSampler<float4,2> pos_buffer;

rtBuffer<float4,2> output0;

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint2, launch_dim, rtLaunchDim, );

```

```

rtDeclareVariable(PerRayDataResult, prdr, rtPayload, );

RTPROGRAM void buffer_camera(void)
{
    float4 i = tex2D( pos_buffer, launch_index.x,
                      launch_index.y );
    PerRayDataResult prdr;
    prdr.result = make_float4(1.0f);
    if (i.w > 0.0f) {
        float3 ray_origin = make_float3(i);
        float3 lDir = make_float3(-lightDir);
        optix::Ray ray = optix::make_Ray(
            ray_origin, lDir, 0, 0.0001f,
            RT_DEFAULT_MAX);
        rtTrace(top_object, ray, prdr);
    }
    else
        prdr.result = make_float4(1.0f);

    //prdr.result.x = ray_origin.x/256.0; prdr.result
    .y = ray_origin.y/256.0; prdr.result.z =
    ray_origin.z/256.0; prdr.result.w = 1.0;
    output0[launch_index] = prdr.result * 0.5 + 0.5;
}

RTPROGRAM void any_hit_shadow()
{
    prdr.result = make_float4(0.0f);
    rtTerminateRay();
}

RTPROGRAM void exception(void)
{
    output0[launch_index] = make_float4(1.f, 0.f, 0.f
                                      , 1.f);
}

RTPROGRAM void geometryintersection(int primIdx)
{
    float4 vecauxa = vertex_buffer[index_buffer[
        primIdx * 3]];
    float4 vecauxb = vertex_buffer[index_buffer[

```

```

        primIdx*3+1]];
float4 vecauxc = vertex_buffer [ index_buffer [
    primIdx*3+2]];
//      float3 e1 , e2 , h , s , q;
//      float a,f,u,v,t;

float3 v0 = make_float3(vecauxa);
float3 v1 = make_float3(vecauxb);
float3 v2 = make_float3(vecauxc);

// Intersect ray with triangle
float3 n;
float t, beta, gamma;
if( intersect_triangle( ray , v0 , v1 , v2 , n , t , beta ,
    gamma ) ) {

    if( rtPotentialIntersection( t ) ) {

/*      float3 n0 = make_float3(normal[ index_buffer [
primIdx*3]]);
      float3 n1 = make_float3(normal[ index_buffer [
primIdx*3+1]]);
      float3 n2 = make_float3(normal[ index_buffer [
primIdx*3+2]]);

      float3 t0 = make_float3(texCoord0[ index_buffer
[ primIdx*3]]);
      float3 t1 = make_float3(texCoord0[ index_buffer
[ primIdx*3+1]]);
      float3 t2 = make_float3(texCoord0[ index_buffer
[ primIdx*3+2]]);

      shading_normal = normalize( n0*(1.0f-beta-gamma)
+ n1*beta + n2*gamma );
      texCoord = t0*(1.0f-beta-gamma) + t1*beta + t2
*gamma ;
      geometric_normal = normalize( n );*/
rtReportIntersection(0);
    }
}
}

RTPROGRAM void boundingbox(int primIdx , float result[6])
{
    float3 v0 = make_float3(vertex_buffer [
        index_buffer [ primIdx*3]]);
    float3 v1 = make_float3(vertex_buffer [
        index_buffer [ primIdx*3+1]]);
```

```

float3 v2 = make_float3( vertex_buffer [
    index_buffer [ primIdx*3+2]]);

const float area = length(cross(v1-v0, v2-v0));

optix::Aabb* aabb = (optix::Aabb*) result;

if(area > 0.0f && !isinf(area)) {
    aabb->m_min = fminf( fminf( v0, v1), v2 )
    ;
    aabb->m_max = fmaxf( fmaxf( v0, v1), v2 )
    ;
}
else {
    aabb->invalidate();
}
}

```

B.2 reflection.cu

```

#include <optix_device.h>
#include <optixu/optixu_math_namespace.h>
#include <optixu/optixu_matrix_namespace.h>
#include <optixu/optixu_aabb.h>
#include <optixu/optixu_aabb_namespace.h>

using namespace optix;

#define MAX_DEPTH 5

struct PerRayDataResult
{
    float3 radiance;
    int depth;
};

rtDeclareVariable(float3, eye, , );
rtDeclareVariable(float4, lightDir, , );
rtDeclareVariable(float3, lightColor, , );
rtDeclareVariable(rtObject, top_object, , );

// Material
rtDeclareVariable(float4, diffuse, , );
rtDeclareVariable(int, texCount, , );
rtTextureSampler<float4, 2> tex0;

rtBuffer<float4> vertex_buffer;
rtBuffer<uint> index_buffer;

```

```

rtBuffer<float4> normal;
rtBuffer<float4> texCoord0;

rtTextureSampler<float4,2> pos_buffer;
rtTextureSampler<float4,2> norm_buffer;
rtTextureSampler<float4,2> color_buffer;

rtBuffer<float4,2> output0;

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint2, launch_dim, rtLaunchDim, );
rtDeclareVariable(PerRayDataResult, prdr, rtPayload, );

rtDeclareVariable(float3, texCoord, attribute texcoord, )
;
rtDeclareVariable(float3, geometric_normal, attribute
    geometric_normal, );
rtDeclareVariable(float3, shading_normal, attribute
    shading_normal, );

rtDeclareVariable(int, Phong, , );

rtDeclareVariable(float, t_hit, rtIntersectionDistance, )
;

__device__ inline float rand(float2 co){
    float intpart;
    return modf(sin(dot(co, make_float2( 12.9898, 78.233))
        ) * 43758.5453,&intpart);
}

RTPROGRAM void buffer_camera(void)
{
    float4 pos = tex2D( pos_buffer, launch_index.x,
        launch_index.y );
    float4 norm = tex2D( norm_buffer, launch_index.x,
        launch_index.y );
    float4 color = tex2D( color_buffer, launch_index.
        x, launch_index.y );
    PerRayDataResult prdr;

    prdr.depth=0;

    if(color.x>0.0f && color.y==0.0f && color.z>0.0
        && color.w==0.0f){
        float3 dir=make_float3(pos) - eye;
        dir=normalize(dir);
    }
}

```

```

        float3 dir_reflect=reflect(dir ,
            make_float3(norm));
        Ray ray = make_Ray(make_float3(pos) ,
            dir_reflect ,Phong ,0.0001f ,
            RT_DEFAULT_MAX);
        rtTrace(top_object ,ray ,prdr);
        output0[launch_index] = make_float4(prdr .
            radiance ,1.f); // (1.f-COEF)*color+COEF*
            prdr.radiance;
    }
    else output0[launch_index]=color;
}

RTPROGRAM void closest_hit(){
    float3 world_geo_normal=normalize(
        rtTransformNormal(RT_OBJECT_TO_WORLD,
            geometric_normal));
    float3 world_shade_normal=normalize(
        rtTransformNormal(RT_OBJECT_TO_WORLD,
            shading_normal));
    float3 ffnormal=faceforward(world_shade_normal ,
        -ray.direction , world_geo_normal);

    float intensity=max(dot(make_float3(-lightDir) ,
        ffnormal) ,0.0f);
    float3 lightIntensityDiffuse=lightColor*intensity;
    float3 tmp_color=make_float3(1.f);

    if(texCount==0){
        tmp_color=make_float3(diffuse)*
            lightIntensityDiffuse + (0.3*
            make_float3(diffuse));
    }
    else{
        tmp_color=(make_float3(diffuse)*
            lightIntensityDiffuse+0.3)*make_float3(
            tex2D(tex0 , texCoord.x , texCoord.y));
    }
    prdr.radiance=tmp_color;
}

RTPROGRAM void closest_hit_glass(){
    if(prdr.depth<MAXDEPTH){
        prdr.depth++;
        float3 world_geo_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
                geometric_normal));
        float3 world_shade_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
                shading_normal));

```

```

        float3 ffnormal=faceforward(
            world_shade_normal , -ray.direction ,
            world_geo_normal);
        float3 dir=reflect(ray.direction,ffnormal
            );
        float3 position = ray.origin + t_hit *
            ray.direction;
        Ray refl=make_Ray(position,dir,Phong
            ,0.0001, RT.DEFAULT.MAX);
        rtTrace(top_object,refl,prdr);
    }
    else{
        prdr.radiance=make_float3(0.f);
    }
}

RT_PROGRAM void miss(){
    prdr.radiance=make_float3(0.0f);
}

RT_PROGRAM void exception(void)
{
    output0[launch_index] = make_float4(1.f, 0.f, 0.f
        , 1.f);
}

RT_PROGRAM void geometryintersection(int primIdx)
{
    float4 vecauxa = vertex_buffer[index_buffer[
        primIdx*3]];
    float4 vecauxb = vertex_buffer[index_buffer[
        primIdx*3+1]];
    float4 vecauxc = vertex_buffer[index_buffer[
        primIdx*3+2]];
//    float3 e1, e2, h, s, q;
//    float a,f,u,v,t;

    float3 v0 = make_float3(vecauxa);
    float3 v1 = make_float3(vecauxb);
    float3 v2 = make_float3(vecauxc);

    // Intersect ray with triangle
    float3 n;
    float t, beta, gamma;
    if( intersect_triangle( ray, v0, v1, v2, n, t, beta,
        gamma ) ) {

        if( rtPotentialIntersection( t ) ) {

```

```

        float3 n0 = make_float3(normal[ index_buffer [
            primIdx*3]] );
        float3 n1 = make_float3(normal[ index_buffer [
            primIdx*3+1]] );
        float3 n2 = make_float3(normal[ index_buffer [
            primIdx*3+2]] );

        float3 t0 = make_float3(texCoord0[ index_buffer
            [ primIdx*3]] );
        float3 t1 = make_float3(texCoord0[ index_buffer
            [ primIdx*3+1]] );
        float3 t2 = make_float3(texCoord0[ index_buffer
            [ primIdx*3+2]] );

        shading_normal = normalize( n0*(1.0f-beta-gamma)
            + n1*beta + n2*gamma );
        texCoord = t0*(1.0f-beta-gamma) + t1*beta + t2
            *gamma ;
        geometric_normal = normalize( n ) ;

        rtReportIntersection(0);
    }
}
}

RT_PROGRAM void boundingbox(int primIdx, float result[6])
{
    float3 v0 = make_float3(vertex_buffer [
        index_buffer [primIdx*3]] );
    float3 v1 = make_float3(vertex_buffer [
        index_buffer [primIdx*3+1]] );
    float3 v2 = make_float3(vertex_buffer [
        index_buffer [primIdx*3+2]] );

    const float area = length(cross(v1-v0, v2-v0));

    optix::Aabb* aabb = (optix::Aabb*)result;

    if(area > 0.0f && !isinf(area)) {
        aabb->m_min = fminf( fminf( v0, v1), v2 )
            ;
        aabb->m_max = fmaxf( fmaxf( v0, v1), v2 )
            ;
    }
    else {
        aabb->invalidate();
    }
}

```

B.3 refraction.cu

```
#include <optix_device.h>
#include <optixu/optixu_math_namespace.h>
#include <optixu/optixu_matrix_namespace.h>
#include <optixu/optixu_aabb.h>
#include <optixu/optixu_aabb_namespace.h>

using namespace optix;

#define COEF 1.1f
#define MAXDEPTH 5

struct PerRayDataResult
{
    float3 radiance;
    int depth;
};

rtDeclareVariable(float3, eye, , );
rtDeclareVariable(float4, lightDir, , );
rtDeclareVariable(float3, lightColor, , );
rtDeclareVariable(rtObject, top_object, , );

// Material
rtDeclareVariable(float4, diffuse, , );
rtDeclareVariable(int, texCount, , );
rtTextureSampler<float4, 2> tex0;

rtBuffer<float4> vertex_buffer;
rtBuffer<uint> index_buffer;
rtBuffer<float4> normal;
rtBuffer<float4> texCoord0;

rtTextureSampler<float4,2> pos_buffer;
rtTextureSampler<float4,2> norm_buffer;
rtTextureSampler<float4,2> color_buffer;

rtBuffer<float4,2> output0;

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint2, launch_dim, rtLaunchDim, );
rtDeclareVariable(PerRayDataResult, prdr, rtPayload, );

rtDeclareVariable(float3, texCoord, attribute texcoord, )
;
```

```

rtDeclareVariable(float3, geometric_normal, attribute
    geometric_normal, );
rtDeclareVariable(float3, shading_normal, attribute
    shading_normal, );

rtDeclareVariable(int, Phong, , );

rtDeclareVariable(float, t_hit, rtIntersectionDistance, )
;

__device__ inline float rand(float2 co){
    float intpart;
    return modf(sin(dot(co, make_float2( 12.9898, 78.233))
        ) * 43758.5453,&intpart);
}

RT_PROGRAM void buffer_camera(void)
{
    float4 pos = tex2D( pos_buffer, launch_index.x,
        launch_index.y );
    float4 norm = tex2D( norm_buffer, launch_index.x,
        launch_index.y );
    float4 color = tex2D( color_buffer, launch_index.
        x, launch_index.y );
    PerRayDataResult prdr;

    if(color.x>0.0f && color.y==0.0f && color.z>0.0
        && color.w==0.0f){
        float3 dir = make_float3(pos) - eye;
        dir=normalize(dir);

        float3 dir_reflect=reflect(dir ,
            make_float3(norm));
        Ray reflected = make_Ray(make_float3(pos)
            ,dir_reflect ,Phong ,0.0001f ,
            RT_DEFAULT_MAX);
        prdr.depth=0;
        rtTrace(top_object ,reflected ,prdr);
        float3 color_reflect = prdr.radiance;
        prdr.radiance=color_reflect;

        float3 dir_refract=make_float3(1.f);
        if(refract(dir_refract ,dir ,make_float3(
            norm) ,COEF)){
            Ray refracted = make_Ray(
                make_float3(pos) ,dir_refract ,
                Phong ,0.001f ,RT_DEFAULT_MAX);
        }
    }
}

```

```

prdr.depth=0;
rtTrace( top_object , refracted , prdr
);
float3 color_refract=prdr.
radianc e;

float coef=1-COEF*COEF*(1-pow( dot
( make_float3(norm),-dir ),2));
output0[launch_index]=make_float4
((coef*color_refract)+((1-coef
)*color_reflect ),1.f);
}
else{
    output0[launch_index] =
    make_float4(prdr.radianc e ,1.f)
;
}
}
else output0[launch_index]=color;
}

RTPROGRAM void closest_hit(){
float3 world_geo_normal=normalize(
    rtTransformNormal(RT_OBJECT_TO_WORLD,
    geometric_normal));
float3 world_shade_normal=normalize(
    rtTransformNormal(RT_OBJECT_TO_WORLD,
    shading_normal));
float3 ffnormal=faceforward(world_shade_normal, -
ray.direction , world_geo_normal);

float intensity=max(dot( make_float3(-lightDir) ,
ffnormal ) ,0.0f );
float3 lightIntensityDiffuse=lightColor*intensity ;
float3 tmp_color=make_float3(1.f);

if (texCount==0){
    tmp_color=make_float3( diffuse )*
        lightIntensityDiffuse + (0.3*
        make_float3( diffuse )) ;
}
else{
    tmp_color=(make_float3( diffuse )*
        lightIntensityDiffuse+0.3)*make_float3(
        tex2D(tex0 , texCoord.x, texCoord.y));
}
prdr.radianc e=tmp_color;
}

```

```

RTPROGRAM void closest_hit_glass(){
    if(prdr.depth<MAXDEPTH){

        prdr.depth++;
        int current_depth=prdr.depth;

        float3 world_geo_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
            geometric_normal));
        float3 world_shade_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
            shading_normal));
        float3 ffnormal=faceforward(
            world_shade_normal , -ray.direction ,
            world_geo_normal);
        float3 position = ray.origin + t_hit *
            ray.direction;

        float3 dir_reflect=reflect(ray.direction ,
            ffnormal);
        Ray refl=make_Ray( position , dir_reflect ,
            Phong,0.0001 , RT_DEFAULT_MAX);
        rtTrace(top_object , refl , prdr);
        float3 color_reflect=0.99*prdr.radiance;

        float3 dir_refract=make_float3(1.f);
        if(refract(dir_refract , ray.direction ,
            ffnormal ,COEF)){
            Ray refracted=make_Ray( position ,
                dir_refract ,Phong,0.0001f ,
                RT_DEFAULT_MAX);
            prdr.depth=current_depth;
            rtTrace(top_object , refracted , prdr
                );
            float3 color_refract=0.99*prdr.
                radiance;

            float coef=1-COEF*COEF*(1-pow( dot
                (ffnormal,-ray.direction) ,2));
            prdr.radiance=color_reflect;//((
                coef*color_refract )+((1-coef)*
                color_reflect));
        }
    }else{
        prdr.radiance=make_float3(0. f);
    }
}

```

```

RTPROGRAM void miss() {
    prdr.radiance=make_float3(0.0f);
}

RTPROGRAM void exception(void)
{
    output0[launch_index] = make_float4(1.f, 0.f, 0.f
        , 1.f);
}

RTPROGRAM void geometryintersection(int primIdx)
{
    float4 vecauxa = vertex_buffer[index_buffer[
        primIdx*3]];
    float4 vecauxb = vertex_buffer[index_buffer[
        primIdx*3+1]];
    float4 vecauxc = vertex_buffer[index_buffer[
        primIdx*3+2]];
    // float3 e1, e2, h, s, q;
    // float a,f,u,v,t;

    float3 v0 = make_float3(vecauxa);
    float3 v1 = make_float3(vecauxb);
    float3 v2 = make_float3(vecauxc);

    // Intersect ray with triangle
    float3 n;
    float t, beta, gamma;
    if( intersect_triangle( ray, v0, v1, v2, n, t, beta,
        gamma ) ) {

        if( rtPotentialIntersection( t ) ) {

            float3 n0 = make_float3(normal[ index_buffer[
                primIdx*3]]);
            float3 n1 = make_float3(normal[ index_buffer[
                primIdx*3+1]]);
            float3 n2 = make_float3(normal[ index_buffer[
                primIdx*3+2]]);

            float3 t0 = make_float3(texCoord0[ index_buffer
                [primIdx*3]]);
            float3 t1 = make_float3(texCoord0[ index_buffer
                [primIdx*3+1]]);
            float3 t2 = make_float3(texCoord0[ index_buffer
                [primIdx*3+2]]);

```

```

        shading_normal = normalize( n0*(1.0f-beta-gamma)
+ n1*beta + n2*gamma );
        texCoord = t0*(1.0f-beta-gamma) + t1*beta + t2
*gamma ;
        geometric_normal = normalize( n );
    }

    rtReportIntersection(0);
}
}

RTPROGRAM void boundingbox(int primIdx, float result[6])
{
    float3 v0 = make_float3(vertex_buffer[
        index_buffer[primIdx*3]]);
    float3 v1 = make_float3(vertex_buffer[
        index_buffer[primIdx*3+1]]);
    float3 v2 = make_float3(vertex_buffer[
        index_buffer[primIdx*3+2]]);

    const float area = length(cross(v1-v0, v2-v0));

    optix::Aabb* aabb = (optix::Aabb*)result;

    if(area > 0.0f && !isinf(area)) {
        aabb->m_min = fminf( fminf( v0, v1), v2 )
        ;
        aabb->m_max = fmaxf( fmaxf( v0, v1), v2 )
        ;
    }
    else {
        aabb->invalidate();
    }
}

```

B.4 ssao.cu

```

#include <optix_device.h>
#include <optixu/optixu_math_namespace.h>
#include <optixu/optixu_matrix_namespace.h>
#include <optixu/optixu_aabb.h>
#include <optixu/optixu_aabb_namespace.h>

using namespace optix;

#define SQRT_SAMPLES 10
#define OCC_DIST 0.5

```

```

#define OCC_INTEN 1.6f

struct PerRayDataResult
{
    int occlusion;
};

//rtDeclareVariable(float4, lightDir, , );
rtDeclareVariable(rtObject, top_object, , );
rtBuffer<float4> vertex_buffer;
rtBuffer<uint> index_buffer;

rtTextureSampler<float4,2> pos_buffer;
rtTextureSampler<float4,2> norm_buffer;

rtBuffer<float4,2> output0;

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint2, launch_dim, rtLaunchDim, );
rtDeclareVariable(PerRayDataResult, prdr, rtPayload, );

__device__ inline float rand(float2 co){
    float intpart;
    return modf(sin(dot(co, make_float2( 12.9898, 78.233))
        ) * 43758.5453,&intpart);
}

RTPROGRAM void buffer_camera(void)
{
    float4 pos = tex2D( pos_buffer, launch_index.x,
        launch_index.y );
    float4 norm = tex2D( norm_buffer, launch_index.x,
        launch_index.y );
    PerRayDataResult prdr;
    int result=SQRT_SAMPLES*SQRT_SAMPLES;
    prdr.occlusion = 0;
    float3 ffnormal =make_float3(norm); // faceforward
    (make_float3(norm),make_float3(pos),
     make_float3(norm));
    //if (pos.w > 0.0f) {
        Onb onb(ffnormal);
        float3 ray_origin = make_float3(pos);
        float3 dir;

```

```

        float inv_sqrt_samples=1/float (
            SQRT_SAMPLES);
        for( int i=0; i<SQRT_SAMPLES; i++){
            for( int j=0; j<SQRT_SAMPLES; j++)
            {
                int a=i+1, b=j+1;
                float rand1=rand(
                    make_float2(pos.x*a,
                    pos.y*b));
                float rand2=rand(
                    make_float2(pos.x*b,
                    pos.z*a));
                float rand3=rand(
                    make_float2(pos.y*a,
                    pos.z*b));
                float rand4=rand(
                    make_float2(rand1*b,
                    rand2*a));
                float rand5=rand(
                    make_float2(rand3*a,
                    rand4*b));

                float u1=(float(i)+rand5)
                    *inv_sqrt_samples;
                float u2=(float(j)+rand4)
                    *inv_sqrt_samples;
                cosine_sample_hemisphere(
                    u1, u2, dir);
                onb.inverse_transform(dir
                );
                optix::Ray ray = optix::
                    make_Ray(ray_origin,
                    dir, 0, 0.0001f,
                    OCC_DIST);
                rtTrace(top_object, ray,
                    prdr);
                result-=prdr.occlusion;
            }
        }

    /*}
    else
        result=0;*/

    float4 out=make_float4(pow(float(result)/
        float(SQRT_SAMPLES*SQRT_SAMPLES),
        OCC_INTEN));
//prdr.result.x = ray_origin.x/256.0; prdr.result
.y = ray_origin.y/256.0; prdr.result.z =
ray_origin.z/256.0; prdr.result.w = 1.0;

```

```

        output0[ launch_index ] = out;

    }

RTPROGRAM void any_hit_shadow()
{
    prdr.occlusion = 1;
    rtTerminateRay();
}

RTPROGRAM void miss()
{
    prdr.occlusion=0;
}

RTPROGRAM void exception(void)
{
    output0[ launch_index ] = make_float4(1.f, 0.f, 0.f
                                         , 1.f);
}

RTPROGRAM void geometryintersection(int primIdx)
{
    float4 vecauxa = vertex_buffer[ index_buffer [
        primIdx * 3]];
    float4 vecauxb = vertex_buffer[ index_buffer [
        primIdx * 3 + 1]];
    float4 vecauxc = vertex_buffer[ index_buffer [
        primIdx * 3 + 2]];
    //    float3 e1, e2, h, s, q;
    //    float a,f,u,v,t;

    float3 v0 = make_float3(vecauxa);
    float3 v1 = make_float3(vecauxb);
    float3 v2 = make_float3(vecauxc);

    // Intersect ray with triangle
    float3 n;
    float t, beta, gamma;
    if( intersect_triangle( ray, v0, v1, v2, n, t, beta,
                           gamma ) ) {

        if( rtPotentialIntersection( t ) ) {

/*      float3 n0 = make_float3(normal[ index_buffer [
        primIdx * 3]]);
```

```

        float3 n1 = make_float3(normal[ index_buffer [
            primIdx*3+1]]);
        float3 n2 = make_float3(normal[ index_buffer [
            primIdx*3+2]]);

        float3 t0 = make_float3(texCoord0[ index_buffer
            [primIdx*3]]);
        float3 t1 = make_float3(texCoord0[ index_buffer
            [primIdx*3+1]]);
        float3 t2 = make_float3(texCoord0[ index_buffer
            [primIdx*3+2]]);

        shading_normal = normalize( n0*(1.0f-beta-gamma)
            + n1*beta + n2*gamma );
        texCoord = t0*(1.0f-beta-gamma) + t1*beta + t2
            *gamma ;
        geometric_normal = normalize( n );*/
    }

    rtReportIntersection(0);
}

}

RTPROGRAM void boundingbox(int primIdx, float result[6])
{
    float3 v0 = make_float3(vertex_buffer [
        index_buffer[primIdx*3]]);
    float3 v1 = make_float3(vertex_buffer [
        index_buffer[primIdx*3+1]]);
    float3 v2 = make_float3(vertex_buffer [
        index_buffer[primIdx*3+2]]);

    const float area = length(cross(v1-v0, v2-v0));

    optix::Aabb* aabb = (optix::Aabb*)result;

    if(area > 0.0f && !isinf(area)) {
        aabb->m_min = fminf( fminf( v0, v1), v2 )
            ;
        aabb->m_max = fmaxf( fmaxf( v0, v1), v2 )
            ;
    }
    else {
        aabb->invalidate();
    }
}

```

B.5 final.cu

```

#include <optix_device.h>
#include <optixu/optixu_math_namespace.h>
#include <optixu/optixu_matrix_namespace.h>
#include <optixu/optixu_aabb.h>
#include <optixu/optixu_aabb_namespace.h>

using namespace optix;

#define COEF 1.1f
#define MAXDEPTH 5

#define SQRT_SAMPLES 10
#define OCC_DIST 0.5
#define OCC_INTEN 1.6f

struct PerRayDataResult
{
    float3 radiance;
    int depth;
};

struct PerRayData_Shadow{
    float val;
    int hit;
};

rtDeclareVariable(float3 , eye , , );
rtDeclareVariable(float4 , lightDir , , );
rtDeclareVariable(float3 , lightColor , , );
rtDeclareVariable(rtObject , top_object , , );

// Material
rtDeclareVariable(float4 , diffuse , , );
rtDeclareVariable(int , texCount , , );
rtTextureSampler<float4 , 2> tex0;

rtBuffer<float4> vertex_buffer;
rtBuffer<uint> index_buffer;
rtBuffer<float4> normal;
rtBuffer<float4> texCoord0;

rtTextureSampler<float4 ,2> pos_buffer;
rtTextureSampler<float4 ,2> norm_buffer;
rtTextureSampler<float4 ,2> color_buffer;

rtBuffer<float4 ,2> output0;

```

```

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint2, launch_dim, rtLaunchDim, );
rtDeclareVariable(PerRayDataResult, prdr, rtPayload, );
rtDeclareVariable(PerRayData_Shadow, prds, rtPayload, );

rtDeclareVariable(float3, texCoord, attribute texcoord, )
;
rtDeclareVariable(float3, geometric_normal, attribute
    geometric_normal, );
rtDeclareVariable(float3, shading_normal, attribute
    shading_normal, );

rtDeclareVariable(int, Phong, , );
rtDeclareVariable(int, Shadow, , );

rtDeclareVariable(float, t_hit, rtIntersectionDistance, )
;

__device__ inline float rand(float2 co){
    float intpart;
    return modf(sin(dot(co, make_float2( 12.9898, 78.233))
        ) * 43758.5453,&intpart);
}

RTPROGRAM void buffer_camera(void)
{
    float4 pos = tex2D( pos_buffer, launch_index.x,
        launch_index.y );
    float4 norm = tex2D( norm_buffer, launch_index.x,
        launch_index.y );
    float4 color = tex2D( color_buffer, launch_index.
        x, launch_index.y );
    PerRayDataResult prdr;
    PerRayData_Shadow prds;

    float3 color_res=make_float3(0.0);

    if(color.x>0.0f && color.y==0.0f && color.z>0.0
        && color.w==0.0f){
        float3 dir = make_float3(pos) - eye;
        dir=normalize(dir);

        float3 dir_reflect=reflect(dir,
            make_float3(norm));
        Ray reflected = make_Ray( make_float3(pos)
            , dir_reflect, Phong, 0.0001f,
            RT_DEFAULT_MAX);
        prdr.depth=0;
    }
}

```

```

rtTrace( top_object , reflected , prdr );
float3 color_reflect = prdr.radiance;
prdr.radiance=color_reflect;

float3 dir_refract=make_float3(1.f);
if( refract( dir_refract , dir , make_float3(
    norm) ,COEF)) {
    Ray refracted = make_Ray(
        make_float3( pos ) , dir_refract ,
        Phong , 0.001f , RT_DEFAULT_MAX );
    prdr.depth=0;
    rtTrace( top_object , refracted , prdr
        );
    float3 color_refract=prdr.
        radiance;

    float coef=1-COEF*COEF*(1-pow( dot
        ( make_float3( norm ) , -dir ) ,2 ) );

    color_res=(coef*color_refract)
        +((1-coef)*color_reflect );
}
else{
    color_res = prdr.radiance;
}

}
else {
    color_res=make_float3( color );

    Ray shadow=make_Ray( make_float3( pos ) ,
        make_float3(-lightDir) ,Shadow ,0.0001f ,
        RT_DEFAULT_MAX );
    rtTrace( top_object , shadow , prds );
    color_res*=prds.val;
}

if (SQRT_SAMPLES) {
    int result=SQRT_SAMPLES*SQRT_SAMPLES;
    Onb onb( make_float3( norm ) );
    float inv_sqrt_samples=1/float(
        SQRT_SAMPLES );
    for (int i=0; i<SQRT_SAMPLES; i++){
        for (int j=0; j<SQRT_SAMPLES; j++)
        {
            int a=i+1, b=j+1;
            float rand1=rand(
                make_float2( pos .x*a ,

```

```

        pos.y*b));
float rand2=rand(
    make_float2(pos.x*b,
    pos.z*a));
float rand3=rand(
    make_float2(pos.y*a,
    pos.z*b));
float rand4=rand(
    make_float2(rand1*b,
    rand2*a));
float rand5=rand(
    make_float2(rand3*a,
    rand4*b));

float u1=(float(i)+rand5)
*inv_sqrt_samples;
float u2=(float(j)+rand4)
*inv_sqrt_samples;

float3 occ_dir;
cosine_sample_hemisphere(
    u1, u2,occ_dir);
onb.inverse_transform(
    occ_dir);
Ray occ_ray = make_Ray(
    make_float3(pos),
    occ_dir,Shadow,0.0001f
    ,OCC_DIST);
rtTrace(top_object,
    occ_ray,prds);
result-=prds.hit;
}

}

float occlusion_level=float(result)/float
(SQRT_SAMPLES*SQRT_SAMPLES);
occlusion_level=pow(occlusion_level,
    OCC_INTEN);
color_res*=occlusion_level;
}

output0[launch_index]=make_float4(color_res,1.0);
}

RTPROGRAM void any_hit_shadow() {
    prds.val=0.5;
    prds.hit=1;
    rtTerminateRay();
}

```

```

RTPROGRAM void any_hit_shadow_glass () {
    prds . val*=0.9;
    prds . hit=0;
    rtIgnoreIntersection ();
}

RTPROGRAM void miss_shadow () {
    prds . val=1.0;
    prds . hit=0;
}

RTPROGRAM void closest_hit () {
    float3 world_geo_normal=normalize(
        rtTransformNormal(RT_OBJECT_TO_WORLD,
        geometric_normal));
    float3 world_shade_normal=normalize(
        rtTransformNormal(RT_OBJECT_TO_WORLD,
        shading_normal));
    float3 ffnormal=faceforward(world_shade_normal ,
        ray . direction , world_geo_normal);
    float3 position=t_hit*ray . direction+ray . origin;

    float intensity=max(dot( make_float3(-lightDir) ,
        ffnormal ),0.0f);
    float3 lightIntensityDiffuse=lightColor*intensity ;
    float3 tmp_color=make_float3(1. f);

    if (texCount==0){
        tmp_color=make_float3( diffuse)*
            lightIntensityDiffuse + (0.3*
            make_float3( diffuse));
    }
    else{
        tmp_color=(make_float3( diffuse)*
            lightIntensityDiffuse+0.3)*make_float3(
            tex2D(tex0 , texCoord.x , texCoord.y));
    }

    prds . val=1;
    Ray shadow=make_Ray(position , make_float3(-
        lightDir ) ,Shadow ,0.0001f ,RT_DEFAULT_MAX );
    rtTrace( top_object ,shadow ,prds );
    tmp_color*=prds . val;

    if (SQRT_SAMPLES){
        int result=SQRT_SAMPLES*SQRT_SAMPLES;
        Onb onb(ffnormal);
        float inv_sqrt_samples=1/float (
            SQRT_SAMPLES);

```

```

        for( int i=0; i<SQRT_SAMPLES; i++){
            for( int j=0; j<SQRT_SAMPLES; j++){
                {
                    int a=i+1, b=j+1;
                    float rand1=rand(
                        make_float2( position .x
                        *a , position .y*b));
                    float rand2=rand(
                        make_float2( position .x
                        *b , position .z*a));
                    float rand3=rand(
                        make_float2( position .y
                        *a , position .z*b));
                    float rand4=rand(
                        make_float2( rand1*b,
                        rand2*a));
                    float rand5=rand(
                        make_float2( rand3*a,
                        rand4*b));

                    float u1=(float (i)+rand5)
                        *inv_sqrt_samples ;
                    float u2=(float (j)+rand4)
                        *inv_sqrt_samples ;

                    float3 occ_dir;
                    cosine_sample_hemisphere(
                        u1 , u2 ,occ_dir);
                    onb.inverse_transform(
                        occ_dir);
                    Ray occ_ray = make_Ray(
                        position ,occ_dir ,
                        Shadow ,0.0001f ,
                        OCC_DIST);
                    rtTrace(top_object ,
                        occ_ray ,prds);
                    result-=prds.hit ;
                }
            }
        }

        float occlusion_level=float (result )/ float
            (SQRT_SAMPLES*SQRT_SAMPLES) ;
        occlusion_level=pow( occlusion_level ,
            OCC_INTEN) ;
        tmp_color*=occlusion_level ;
    }

    prdr.radiance=tmp_color ;
}

```

```

RTPROGRAM void closest_hit_glass(){
    if(prdr.depth<MAXDEPTH){

        prdr.depth++;
        int current_depth=prdr.depth;

        float3 world_geo_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
            geometric_normal));
        float3 world_shade_normal=normalize(
            rtTransformNormal(RT_OBJECT_TO_WORLD,
            shading_normal));
        float3 ffnormal=faceforward(
            world_shade_normal , -ray.direction ,
            world_geo_normal);
        float3 position = ray.origin + t_hit *
            ray.direction;

        float3 dir_reflect=reflect(ray.direction ,
            ffnormal);
        Ray refl=make_Ray( position , dir_reflect ,
            Phong,0.0001 , RT_DEFAULT_MAX);
        rtTrace(top_object , refl , prdr);
        float3 color_reflect=0.99*prdr.radiance;

        float3 dir_refract=make_float3(1.f);
        if(refract(dir_refract , ray.direction ,
            ffnormal ,COEF)){
            Ray refracted=make_Ray( position ,
                dir_refract ,Phong,0.0001f ,
                RT_DEFAULT_MAX);
            prdr.depth=current_depth;
            rtTrace(top_object , refracted , prdr
                );
            float3 color_refract=0.99*prdr.
                radiance;

            float coef=1-COEF*COEF*(1-pow( dot
                (ffnormal,-ray.direction) ,2));
            prdr.radiance=color_reflect;//((
                coef*color_refract )+((1-coef)*
                color_reflect));
        }
    }else{
        prdr.radiance=make_float3(0. f);
    }
}

```

```

RTPROGRAM void miss() {
    prdr.radiance=make_float3(0.0f);
}

RTPROGRAM void exception(void)
{
    output0[launch_index] = make_float4(1.f, 0.f, 0.f
        , 1.f);
}

RTPROGRAM void geometryintersection(int primIdx)
{
    float4 vecauxa = vertex_buffer[index_buffer[
        primIdx*3]];
    float4 vecauxb = vertex_buffer[index_buffer[
        primIdx*3+1]];
    float4 vecauxc = vertex_buffer[index_buffer[
        primIdx*3+2]];
    // float3 e1, e2, h, s, q;
    // float a,f,u,v,t;

    float3 v0 = make_float3(vecauxa);
    float3 v1 = make_float3(vecauxb);
    float3 v2 = make_float3(vecauxc);

    // Intersect ray with triangle
    float3 n;
    float t, beta, gamma;
    if( intersect_triangle( ray, v0, v1, v2, n, t, beta,
        gamma ) ) {

        if( rtPotentialIntersection( t ) ) {

            float3 n0 = make_float3(normal[ index_buffer[
                primIdx*3]]);
            float3 n1 = make_float3(normal[ index_buffer[
                primIdx*3+1]]);
            float3 n2 = make_float3(normal[ index_buffer[
                primIdx*3+2]]);

            float3 t0 = make_float3(texCoord0[ index_buffer
                [primIdx*3]]);
            float3 t1 = make_float3(texCoord0[ index_buffer
                [primIdx*3+1]]);
            float3 t2 = make_float3(texCoord0[ index_buffer
                [primIdx*3+2]]);

```

```

        shading_normal = normalize( n0*(1.0f-beta-gamma)
+ n1*beta + n2*gamma );
        texCoord = t0*(1.0f-beta-gamma) + t1*beta + t2
*gamma ;
        geometric_normal = normalize( n );
    }

    rtReportIntersection(0);
}
}

RTPROGRAM void boundingbox(int primIdx, float result[6])
{
    float3 v0 = make_float3(vertex_buffer[
        index_buffer[primIdx*3]]);
    float3 v1 = make_float3(vertex_buffer[
        index_buffer[primIdx*3+1]]);
    float3 v2 = make_float3(vertex_buffer[
        index_buffer[primIdx*3+2]]);

    const float area = length(cross(v1-v0, v2-v0));

    optix::Aabb* aabb = (optix::Aabb*)result;

    if(area > 0.0f && !isinf(area)) {
        aabb->m_min = fminf( fminf( v0, v1), v2 )
        ;
        aabb->m_max = fmaxf( fmaxf( v0, v1), v2 )
        ;
    }
    else {
        aabb->invalidate();
    }
}

```