PBRTv4-小专题-PBRT中的可交互渲染流程

Dezeming Family

2023年8月31日

DezemingFamily系列文章和电子文档**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了DezemingFamily的系列文章，可以从我们的网站[https://dezeming.top/]找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

目录

一 GUI

GUI就是GLFW窗口，见pbrt/util/gui.h和gui.cpp文件。

1 1 GUI类与交互

该类的初始化构造函数需要窗口名title，显示分辨率resolution和场景边界sceneBounds。构造函数把GLFW环境初始化好以后，根据是CPU渲染还是GPU渲染选择是初始化cudaFramebuffer还是cpuFramebuffer。

WavefrontPathIntegrator::Render()函数会创建并初始化GUI，然后，当渲染完每帧图像以后，会调用UpdateFilm()函数更新film对象。

GUI::RefreshDisplay()函数中会根据设置的环境选择刷新方式。

```
if (Options->useGPU)
    cudaFramebuffer->Draw(width, height);
else {
    GL_CHECK(glEnable(GL_FRAMEBUFFER_SRGB));
    GL_CHECK(glRasterPos2f(-1, 1));
    GL_CHECK(glPixelZoom(pixelScales[0], -pixelScales[1]));
    GL_CHECK(
        glDrawPixels(resolution.x, resolution.y, GL_RGB, GL_FLOAT,
            cpuFramebuffer));
}
```

该函数会根据一些状态返回一些值，比如：

```
if (glfwWindowShouldClose(window))
    return DisplayState::EXIT;
else if (process())
    return DisplayState::RESET;
else
    return DisplayState::NONE;
```

只要有相关按键按下或鼠标按下并移动，GUI::process()就会返回true值。在下一轮渲染中，

WavefrontPathIntegrator类的ParallelFor(...)函数根据是否调用GPU来选择是CPU还是GPU渲染方式。有很多函数都会调用ParallelFor(...)函数，比如光线的生成、光线和物体的求交、追踪阴影光线等。

WavefrontPathIntegrator::Render()的逻辑是这样的：

```
for (int sampleIndex = firstSampleIndex; sampleIndex < lastSampleIndex ||
    gui; ++sampleIndex) {
    if (sampleIndex < lastSampleIndex) {
        for (int y0 = pixelBounds.pMin.y; y0 < pixelBounds.pMax.y; y0 +=
            scanlinesPerPass) {
            GenerateCameraRays(...);
            for (int wavefrontDepth = 0; true; ++wavefrontDepth) {
                // rendering
                ......
            }
        }
    }
    if (gui) {
```

```
        // put the buffer in film to gui to display
        RGB *rgb = gui->MapFramebuffer();
        UpdateFramebufferFromFilm(pixelBounds, gui->exposure, rgb);
        gui->UnmapFramebuffer();
        if (state == DisplayState::EXIT)
            break;
        else if (state == DisplayState::RESET) {
            // reset the pixel buffer in film
            ......
        }
    }
}
```

关于相机交互后缓冲区的更新，见WavefrontPathIntegrator::Render()函数：

```
if (state == DisplayState::RESET) {
    sampleIndex = firstSampleIndex - 1;
    ParallelFor(
    "Reset pixels", resolution.x * resolution.y,
    PBRT_CPU_GPU_LAMBDA(int i) {
        int x = i % resolution.x, y = i / resolution.x;
        film.ResetPixel(pixelBounds.pMin + Vector2i(x, y));
    });
}
```

ParallelFor(...)函数第三个参数接受一个模板，这个模板在pbrt中是一个函数：

```
template <typename F>
void ParallelFor(const char *description, int nItems, F &&func) {
    if (Options->useGPU)
        GPUParallelFor(description, nItems, func);
    else
        pbrt::ParallelFor(0, nItems, func);
}
```

1 2 键盘控制

键盘控制有下面这些选项：

- w, a, s, d: move the camera forward and back, left and right.
- q, e: move the camera down and up, respectively.
- Arrow keys: adjust the camera orientation.
- B, b: respectively increase and decrease the exposure (”brightness”).
- c: print the transformation matrix for the current camera position.
- -, =: respectively decrease and increase the rate of camera movement.

二 Frame Buffer

下面的代码将film的buffer转移到gui的buffer上：

```
RGB *rgb = gui->MapFramebuffer();
UpdateFramebufferFromFilm(pixelBounds, gui->exposure, rgb);
gui->UnmapFramebuffer();
```

在CUDAOutputBuffer的初始化时，cudaGraphicsResource对象会和OpenGL的PBO绑定，作为图形资源。GUI::MapFramebuffer()就相当于获得Framebuffer的指针。然后使用UpdateFramebufferFromFilm(...)往里面填充数据，该函数给每个像素的float类型的rgb值都乘以exposure值。

之后，会调用GUI::RefreshDisplay()来刷新。该函数调用CUDAOutputBuffer::Draw(..)来绘制。Draw(..)会调用BufferDisplay::display(...)，注意该函数开启了GL_FRAMEBUFFER_SRGB，OpenGL将自动执行gamma校正。

关于OpenGL和CUDA的互操作相关内容可以去查找CUDA手册来辅助阅读，这里不再花过多篇幅去讲解。

三 G-Buffer

PBRT v4的特点之一就是可以获得G-Buffer并保存。

3 1 保存G-Buffer

在场景文件.pbrt文件中，Film一般都是"rgb"。如果是"gbuffer"，就会创建GBufferFilm。

注意修改.pbrt文件时，filename要改成.exr，因为只支持.exr文件。

```
Film "gbuffer"
    "string filename" [ "living-room.exr" ]
```

保存的结果是一个有着很多通道的.exr文件。

注意.exr文件默认保存为16位的浮点数：

```
bool writeFP16 = parameters.GetOneBool("savefp16", true);
```

如果这样设置就可以保存为32位浮点数：

```
Film "gbuffer"
    "string filename" [ "living-room.exr" ]
        "bool savefp16" [ false ]
```

位置、法线和屏幕空间z导数在默认情况下在相机空间中。或者，GBufferFilm的“coordinatesystem”参数可以用于指定世界空间输出：

```
Film "gbuffer"
    "string filename" [ "living-room.exr" ]
        "bool savefp16" [ false ]
        "string coordinatesystem" [ "world" ]
```

3 2 获得渲染中的G-Buffer

GBufferFilm::AddSample(...)中把每个像素的visibleSurface获得的内容填充到Array2D¡Pixel¿pixels中。

四　渲染的执行与更新Film

见WavefrontPathIntegrator::Render()函数的下面的代码块：

```
for (int sampleIndex = firstSampleIndex; sampleIndex < lastSampleIndex ||
    gui; ++sampleIndex) {
    if (sampleIndex < lastSampleIndex) {
        for (int y0 = pixelBounds.pMin.y; y0 < pixelBounds.pMax.y; y0 +=
            scanlinesPerPass) {
            GenerateCameraRays(...);
            for (int wavefrontDepth = 0; true; ++wavefrontDepth) {
                Reset queues
                GenerateRaySamples(...)
                Find closest intersections
                SampleMediumInteraction(.)
                HandleEscapedRays()
                HandleEmissiveIntersection()
                if (wavefrontDepth == maxDepth)
                    break;
                EvaluateMaterialsAndBSDFs(..)
                TraceShadowRays(.)
                SampleSubsurface(.)
            }
            UpdateFilm();
        }
        Copy updated film pixels to buffer for the display server
    }
    if (gui) {...}
}
```

每个扫描线的内容渲染完以后，就会调用UpdateDisplayRGBFromFilm(.)函数将Film里的内容复制到GUI显示的缓冲区中。

我不清楚为什么每行扫描线扫面完就更新Film，其实经过实验，把整幅图像都渲染完一spp以后再更新Film也是完全可以的。这可能是PBRT的一点小BUG吧。

五　小结

本文介绍了PBRT v4中可交互渲染的一些相关构成。但本文不一定是最终版本，后面可能遇到想放到本文中的内容还会再进行补充。如果您获得了DezemingFamily的系列文章，可以从我们的网站[https://dezeming.top/]找到最新的版本。

参考文献

[1] https://github.com/mmp/pbrt-v4

[2] https://pbrt.org/

[3] https://developer.nvidia.com/designworks/optix/downloads/legacy

[4] https://pbrt.org/resources

[5] https://www.cnblogs.com/Heskey0/category/2166679.html

[6] https://github.com/ebruneton/clear-sky-models/tree/master/atmosphere/model/hosek

[7] https://www.cnblogs.com/Heskey0/p/15973546.html