OIT（Order Independent Transparency，顺序无关透明）

This file is a document to explain how to do the Order Independent Transparency.

Alpha 通道

Porter 在 1984 年提出了 Alpha 通道（1.[Porter 1984]），目前在实时渲染中已被广泛地用于模拟物体的透明效果。

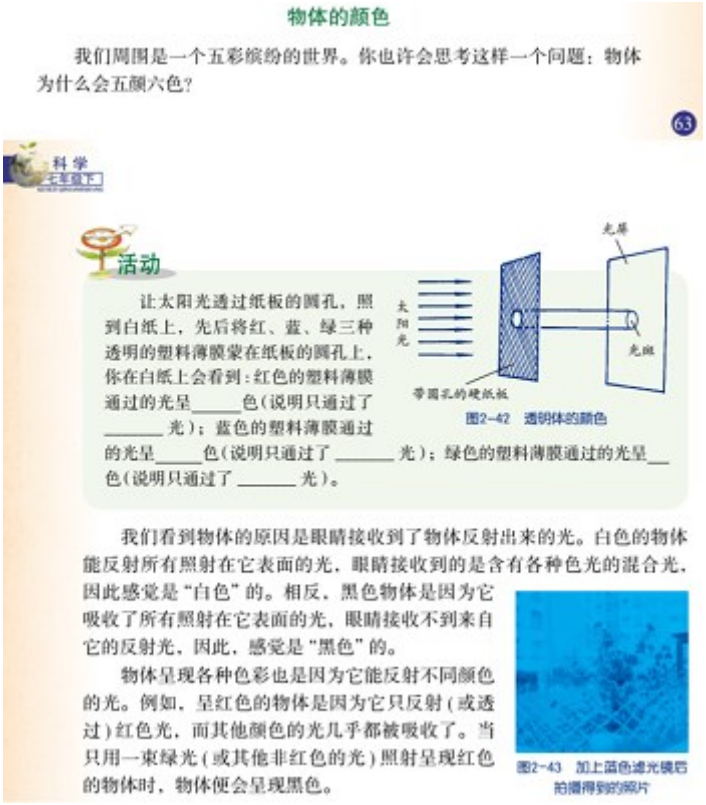
我们设对应到同一个像素（Pixel）的一系列片元（Fragment）的颜色、Alpha、深度为三元组$[C_i\ A_i\ Z_i]$，该像素最终的颜色$C_{Final}=\sum_i\left(\left(\prod_{z_j\ Nearer\ z_i}(1-A_j)\right)A_iC_i\right)$ //其中，$V(z_i)=\prod_{z_j\ Nearer\ z_i}(1-A_j)$又被称为可见性函数，即$C_{Final}=\sum_i(V(z_i)A_iC_i)$ 。

值得注意的是，在物理含义上， Alpha 模拟的是局部覆盖（Partial Coverage）而非透射率（Transmittance）。

Alpha 的含义是片元覆盖的面积占像素面积的比例（这也是我们用标量 float 而非向量 RGB 来表示 Alpha 的原因；这种情况在一些文献中被称作波长无关的（Wavelength-Independent））。比如，我们透过一条蓝色的真丝围巾观察一块红色的砖，我们看到砖的颜色大体为蓝色和红色“相加”； 真丝围巾的纤维本身是不透明的，只是真丝围巾的纤维之间存在着间隙，我们通过这些间隙看到了红色的砖，即真丝围巾“局部覆盖”了砖。

而透射率是波长相关的（Wavelengh-Dependent）；比如，我们透过一块蓝色的塑料薄膜观察一块红色的砖，我们看到的砖的颜色大体为黑色（即蓝色和红色“相乘”）；红色的砖只

反射红色的光，而蓝色的塑料薄膜只允许蓝色的光通过，红色的砖的反射光全部会被蓝色的塑料薄膜吸收，即呈现出黑色（参考文献：[《科学七年级下册》（ISBN: 9787553603162）/第 2 章对环境的感觉/第 4 节光的颜色/物体的颜色]）；如果需要模拟透射率相关的效果，那么我们应当使用参与介质（Participating Media）（2.[Yusor 2013]、3.[Hoobler 2016]）相关的技术。



物体的颜色

我们周围是一个五彩缤纷的世界。你也许会思考这样一个问题：物体为什么会五颜六色？

63

科学 七年级下

活动

让太阳光透过纸板的圆孔，照到白纸上，先后将红、蓝、绿三种透明的塑料薄膜蒙在纸板的圆孔上，你在白纸上会看到：红色的塑料薄膜通过的光呈_____色（说明只通过了_____光）；蓝色的塑料薄膜通过的光呈_____色（说明只通过了_____光）；绿色的塑料薄膜通过的光呈_____色（说明只通过了_____光）。

太阳光 光屏 光斑 带圆孔的硬纸板

图2-42 透明体的颜色

我们看到物体的原因是眼睛接收到了物体反射出来的光。白色的物体能反射所有照射在它表面的光，眼睛接收到的是含有各种色光的混合光，因此感觉是“白色”的。相反，黑色物体是因为它吸收了所有照射在它表面的光，眼睛接收不到来自它的反射光，因此，感觉是“黑色”的。

物体呈现各种色彩也是因为它能反射不同颜色的光。例如，呈红色的物体是因为它只反射（或透过）红色光，而其他颜色的光几乎都被吸收了。当只用一束绿光（或其他非红色的光）照射呈现红色的物体时，物体便会呈现黑色。

图2-43 加上蓝色滤光镜后拍摄得到的照片

图来自：[《科学七年级下册》（ISBN: 9787553603162）/第 2 章对环境的感觉/第 4 节光的颜色/物体的颜色]

根据 Alpha 的含义，不难理解可见性函数$V(z_i) = \prod_{z_j \text{ Nearer } z_i}(1 - A_j)$，即只有比当前片元“更近”（Nearer）的片元才会局部覆盖当前片元。（一些文献中将可见性函数 $V(Z_i)$称作透射率 $T(Z_i)$，在严格意义上是错误的。）

顺序性透明

在实时渲染中，比较常见的做法是将几何体排序后用 Over/Under 操作（1.[Porter 1984]、4. [Dunn 2014]）以递归的方式求解 C_{Final}：

1.OpaquePass 渲染不透明物体，得到 BackgroundColor 和 BackgroundDepth。

2.TransparencyPass 将 BackgroundDepth 用于深度测试（关闭深度写入）将透明物体从后往前/从前往后排序后用 Over/Under 操作以递归的方式求解 C_{Final}。

Over 操作

将几何体从后往前排序后用 Over 操作以递归的方式求解 C_{Final}

C_{Final_0} = BackgroundColor

$C_{Final\_n} = (A_nC_n) + (1-A_n)C_{Final\_n-1}$

Under 操作

将几何体从前往后排序后用 Under 操作以递归的方式求解 C_{Final}

$C_{Final\_0} = 0$

$A_{Total\_0} = 1$

$C_{Final\_n} = A_{Total\_n-1}(A_nC_n) + C_{Final\_n-1}$

$A_{Total\_n} = A_{Total\_n-1}(1-A_n)$ //A_{Total} 即可见性函数 $V(Z_i)$

OpaquePass 得到的图像将在最后以 A = 1, C = BackgroundColor 的形式合成上去

在严格意义上，只有将片元从后往前/从前往后排序，才能保证 Over/Under 操作的正确性。而在实时渲染中，排序的粒度是基于物体而非基于片元；如果物体内部存在穿插，那么片元的顺序将不符合从后往前/从前往后，从而导致 Over/Under 操作的结果存在错误。因此，人们不得不探索 OIT 算法来解决这个问题。

深度剥离（Depth Peeling）

深度剥离（5.[Everitt 2001]）是一种比较古老的在实时渲染中被实际应用的 OIT 算法。

1.OpaquePass 渲染不透明物体，得到 BackgroundColor 和 BackgroundDepth。

2.NearestLayerPass 将 Depth 初始化为 BackgroundDepth 开启深度测试和深度写入 将透明物体从前往后排序后绘制得到 NearestLayerColor 和 NearestLayerDepth 并用 Under 操作将 NearestLayerColor 合成到 C_{Final} //注：深度剥离本身并不依赖于片元的顺序，之所以排序是为了充分发挥硬件的 EarlyDepthTest 来提升性能

3.SecondNearestLayerPass 将 Depth 初始化为 BackgroundDepth 开启深度测试和深度写入 将 NearestLayerDepth 绑定到片元着色器的纹理单元并在片元着色器中显式 Discard 掉 Depth NearerOrEqual NearestLayerDepth 的片元 将透明物体从前往后排序后绘制得到 SecondNearestLayerColor 和 SecondNearestLayerDepth 并用 Under 操作将 SecondNearestLayerColor 合成到 C_{Final}

4.ThirdNearestLayerPass 将 Depth 初始化为 BackgroundDepth 开启深度测试和深度写入 将 SecondNearestLayerDepth 绑定到片元着色器的纹理单元 以此类推...

... // N 个 Pass 可以剥离得到 N 个最近的层，应用程序可以根据自身的需求选择 Pass 的个数

最后用 Under 操作将 OpaquePass 得到的 BackgroundColor 合成到 C_{Final}

在理论上，深度剥离也可以从远到近剥离各层，并用 Over 操作合成到 C_{Final}。只不过，在 Under 操作中，如果应用程序选择的 Pass 个数过低，不能剥离得到所有的层，那么最远处的若干层会被忽略；由于 A_{Total}（即可见性函数 $V(Z_i)$）是单调递减的，最远处的若干层对 C_{Final} 的贡献是较低的，产生的误差也是较低的。这也是深度剥离采用 Under 操作而非 Over 操作的原因。

显然，深度剥离有一个显著的缺陷——Pass 个数过多——在效率上存在着比较严重的问题；因此在被提出以后的数十年间并没有流行起来。

随机透明（Stochastic Transparency）

$C_{Final} = \sum_i (V(z_i) A_i C_i)$的求解依赖于片元的顺序，很大程度上是由于可见性函数$V(z_i) = \prod_{Z_j \text{ Nearer } Z_i} (1 - A_j)$依赖于片元的顺序。随机透明（6.[Enderton 2010]）基于概率论的原理，利用硬件的 MSAA 特性进行随机抽样，提出了一种顺序无关的求解可见性函数 $V(Z_i)$的方式。

随机深度（Stochastic Depth）

我们假设：通过设置 gl_SampleMask[]/SV_Coverage 的值，以确保片元[C_i A_i Z_i]在生成采样点[Z_i]时，每个采样点被覆盖的概率为 A_i；开启深度测试和深度写入，以确保较近的片元生成的采样点一定覆盖较远的片元生成的采样点；不同片元生成采样点时，每个采样点被覆盖的概率相互独立（Uncorrelated）；那么在最终生成的 Depth 图像中，对任意采样点[Z]，满足 $Z_i \leqslant Z$ 的概率即为可见性函数$V(z_i) = \prod_{Z_j \text{ Nearer } Z_i} (1 - A_j)$。

可以根据概率论的相关知识证明：满足 $Z_i \leqslant Z$ 即采样点[Z]被片元[C_i A_i Z_i]或被比片元[C_i A_i Z_i]更远的片元覆盖，即采样点[Z]不被比片元[C_i A_i Z_i]更近的片元[C_j A_j Z_j]覆盖，由于采样点[Z]不被比片元[C_i A_i Z_i]更近的片元[C_j A_j Z_j]覆盖的概率为 1 - A_j，且概率之间相互独立，最终的概率为各概率相乘，即$\prod_{Z_j \text{ Nearer } Z_i} (1 - A_j)$。

我们设，在 MSAA 中，1 个片元对应的采样点个数为 S，满足 $Z_i \leqslant Z$ 的采样点的个数为 $Count(Z_i)$，$SV(Z_i) = Count(Z_i) / S$；不难证明 $SV(Z_i)$的数学期望为 $V(Z_i)$，$Count(Z_i)$可以通过纹理采样得到，可以用 $SV(Z_i)$近似地表示可见性函数 $V(Z_i)$。

Alpha 校正（Alpha Correction）

不难证明，$\sum_i (V(z_i) A_i) = 1 - \prod_i (1 - A_i)$

不妨设 $A_0 = 0.4$ $A_1 = 0.7$ $A_2 = 0.6$，我们有：

左边 = 0.4 + (1 - 0.4)×0.7 + (1 - 0.4)×(1 - 0.7)×0.6

= 0.4 + (1 - 0.4)×0.7 + (1 - 0.4)×(1 - 0.7)×0.6 + (1 - 0.4)×(1 - 0.7)×(1 - 0.6) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 0.4 + (1 - 0.4)×0.7 + (1 - 0.4)×(1 - 0.7)×0.6 + (1 - 0.4)×(1 - 0.7)×(1 - 0.6) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 0.4 + (1 - 0.4)×0.7 + (1 - 0.4)×(1 - 0.7)×(0.6 + 1 - 0.6) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 0.4 + (1 - 0.4)×0.7 + (1 - 0.4)×(1 - 0.7) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 0.4 + (1 - 0.4)×(0.7 + 1 - 0.7) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 0.4 + (1 - 0.4) - (1 - 0.4)×(1 - 0.7)×(1 - 0.6)

= 1 - (1 - 0.4)×(1 - 0.7)×(1 - 0.6) = 右边

不难证明，$\sum_i (SV(z_i) A_i)$的数学期望为$\sum_i (V(z_i) A_i)$即$1 - \prod_i (1 - A_i)$

Alpha 校正可以认为是对 SV(Z_i)进行归一化（Normalize），即假定$\frac{SV(z_i)}{\sum_i(SV(z_i)A_i)}=\frac{V(z_i)}{\sum_i(V(z_i)A_i)}$，从而得到$V(z_i)=SV(z_i)\frac{\sum_i(V(z_i)A_i)}{\sum_i(SV(z_i)A_i)}=SV(z_i)\frac{1-\prod_i(1-A_i)}{\sum_i(SV(z_i)A_i)}$

Render Pass

1.OpaquePass

渲染不透明物体，得到 BackgroundColor 和 BackgroundDepth

2.StochasticDepthPass

将 Depth 初始化为 BackgroundDepth 开启 MSAA 开启深度测试（NearerOrEqual）和深度写入 用伪随机函数基于片元[C_i A_i Z_i]的 A_i 生成 gl_SampleMask[]/SV_Coverage 的值 将透明物体从前往后排序后绘制得到 StochasticDepth //注：随机透明本身并不依赖于片元的顺序，之所以排序是为了充分发挥硬件的 EarlyDepthTest 来提升性能

值得注意的是：

1.StochasticDepthPass 开启的 MSAA 是用于随机抽样的，随机透明本身并不要求除 StochasticDepthPass 以外的 Pass 开启 MSAA；如果应用程序有空间性反走样（Spatial AntiAliasing）的需求，那么可以在除 StochasticDepthPass 以外的 Pass 开启 MSAA（当然，也可以使用其它的空间性反走样算法（比如：FXAA））；除 StochasticDepthPass 以外的 Pass 用于空间反走样的 MSAA 和 StochasticDepthPass 用于随机抽样的 MSAA 并没有任何关系，比如：允许在用于空间反走样的 MSAA 是 4X 的同时，用于随机抽样的 MSAA 为 8X。

2.为了确保采样点被片元覆盖的概率相互独立，必须用伪随机函数基于片元[C_i A_i Z_i]的 A_i 生成 gl_SampleMask[]/SV_Coverage 的值，而不得使用硬件的 AlphaToCoverage 特性。

3.在计算 SV(Z_i)时，Z_i 为着色点的深度；为了保持一致，应当在片元着色器中将着色点的深度写入到 gl_FragDepth/SV_Depth（在默认情况下，采样点的深度会被写入到最终生成的 Depth 图像中）。

4.由于硬件的限制，MSAA 最多为 8X，即 1 个片元对应的采样点个数为 8；在理论上，可以使用多个 Pass 来模拟更多的采样点，但是出于效率的原因，往往只使用 1 个 Pass。

3.AccumulateAndTotalAlphaPass

将 Depth 初始化为 BackgroundDepth 开启深度测试关闭深度写入 开启 MRT 和 Sperable

未完待续

权重融合（Weighted Blended）

K-Buffer

参考文献

1.[Porter 1984] Thomas Porter, Tom Duff. "Compositing Digital Images." SIGGRAPH 1984.
https://keithp.com/~keithp/porterduff/p253-porter.pdf

2.[Yusor 2013] Egor Yusor. "Practical Implementation of Light Scattering Effects Using Epipolar Sampling and 1D Min/Max Binary Trees." GDC 2013.
https://software.intel.com/en-us/blogs/2013/03/18/gtd-light-scattering-sample-updated
https://software.intel.com/en-us/blogs/2013/06/26/outdoor-light-scattering-sample
https://software.intel.com/en-us/blogs/2013/09/19/otdoor-light-scattering-sample-update

3.[Hoobler 2016] Nathan Hoobler. "Fast, Flexible, Physically-Based Volumetric Light Scattering." GDC 2016.
http://developer.nvidia.com/VolumetricLighting

4.[Dunn 2014] Alex Dunn. "Transparency (or Translucency) Rendering." NVIDIA GameWorks Blog 2014.
https://developer.nvidia.com/content/transparency-or-translucency-rendering

5.[Everitt 2001] Cass Everitt. "Interactive Order-Independent Transparency." NVIDIA WhitePaper 2001.
https://www.nvidia.com/object/Interactive_Order_Transparency.html

6.[Enderton 2010] Eric Enderton, Erik Sintorn, Peter Shirley, David Luebke. "Stochastic Transparency." I3D 2010.
https://research.nvidia.com/publication/stochastic-transparency
NVIDIA SDK11 Samples / StochasticTransparency
http://developer.nvidia.com/dx11-samples

7.[Laine 2011] Samuli Laine, Tero Karras. "Stratified Sampling for Stochastic Transparency." EGSR 2011.
https://research.nvidia.com/publication/stratified-sampling-stochastic-transparency

8.[McGuire 2011] Morgan McGuire, Eric Enderton. "Colored Stochastic Shadow Maps". I3D 2011.
http://research.nvidia.com/publication/colored-stochastic-shadow-maps

9.[McGuire 2013] Morgan McGuire, Louis Bavoil. "Weighted Blended Order-Independent Transparency. " JCGT 2013.
http://jcgt.org/published/0002/02/09/

NVIDIA GameWorks Vulkan and OpenGL Samples / Weighted Blended Order-independent Transparency
https://github.com/NVIDIAGameWorks/GraphicsSamples/tree/master/samples/gl4-kepler/WeightedBlendedOIT

10.[McGuire 2016] Morgan McGuire, Michael Mara. "A Phenomenological Scattering Model for Order-Independent Transparency". I3D 2016.
http://research.nvidia.com/publication/phenomenological-scattering-model-order-independent-transparency

11.[McGuire 2017] Morgan McGuire, Michael Mara. "Phenomenological Transparency". TVCG 2017.
http://research.nvidia.com/publication/phenomenological-transparency

12.[Carpenter 1984] Loren Carpenter. "The A-buffer, an Antialiased Hidden Surface Method." SIGGRAPH 1984.
https://dl.acm.org/citation.cfm?id=808585

13.[Bavoil 2007] Louis Bavoil, Steven Callahan, Aaron Lefohn, Joao Comba, Claudio Silva. "Multi-Fragment Effects on the GPU using the k-Buffer." I3D 2007.
https://i3dsymposium.github.io/2007/papers.html

14.Rasterizer Order Views (Direct3D) / Fragment Shader Interlock (OpenGL) / Raster Order Groups (Metal)
Rasterizer Order Views 101: a Primer
https://software.intel.com/en-us/gamedev/articles/rasterizer-order-views-101-a-primer
Programmable Blend with Pixel Shader Ordering
https://software.intel.com/en-us/articles/programmable-blend-with-pixel-shader-ordering
Metal 2 on A11 - Raster Order Groups
https://developer.apple.com/videos/play/tech-talks/605/
Rasterizer Order Views (Direct3D11)
https://docs.microsoft.com/en-us/windows/desktop/direct3d11/rasterizer-order-views
ARB_fragment_shader_interlock (OpenGL)
https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_fragment_shader_interlock.txt
About Raster Order Groups (Metal)
https://developer.apple.com/documentation/metal/mtldevice/ios_and_tvos_devices/about_gpu_family_4/about_raster_order_groups

15.Programmable Blending
EXT_shader_framebuffer_fetch (OpenGL)
https://www.khronos.org/registry/OpenGL/extensions/EXT/EXT_shader_framebuffer_fetch.txt
Subpass Input (Vulkan)
https://github.com/KhronosGroup/GLSL/blob/master/extensions/khr/GL_KHR_vulkan_glsl.txt
Programmable Blending (Metal) "The feature that allows a TBDR GPU to read from attached render

targets in a fragment function is also known as programmable blending."
https://developer.apple.com/documentation/metal/deferred_lighting

16.[Salvi 2010] Marco Salvi,Kiril Vidimce, Andrew Lauritzen, Aaron Lefohn. "Adaptive Volumetric Shadow Maps." EGSR 2010.
https://software.intel.com/en-us/articles/adaptive-volumetric-shadow-maps

17.[Salvi 2011] Marco Salvi, Jefferson Montgomery, Aaron Lefohn. "Adaptive Transparency." High Performance Graphics 2011.
https://software.intel.com/en-us/articles/adaptive-transparency-hpg-2011
Order-Independent Transparency Approximation with Raster Order Views (Update 2017)
https://software.intel.com/en-us/articles/oit-approximation-with-pixel-synchronization-update-2014

18.[Salvi 2014] Marco Salvi, Karthik Vaidyanathan. "Multi-layer Alpha Blending." I3D 2014.
https://software.intel.com/en-us/articles/multi-layer-alpha-blending