中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

多核程序设计与实践

OpenGL与CUDA

陶钧

taoj23@mail.sysu.edu.cn

中山大学 数据科学与计算机学院

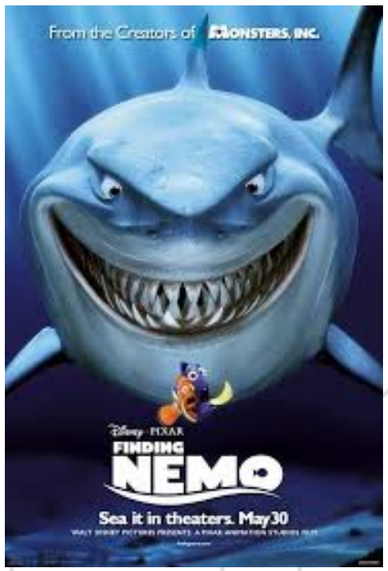
国家超级计算广州中心

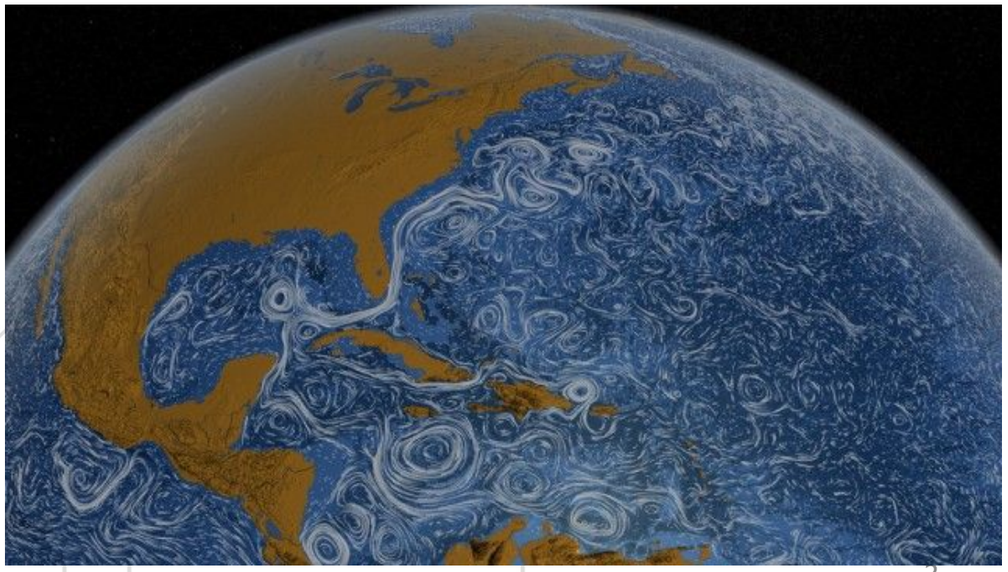
课程提纲

- OpenGL简介
- CUDA与OpenGL交互
- 应用举例

OpenGL简介

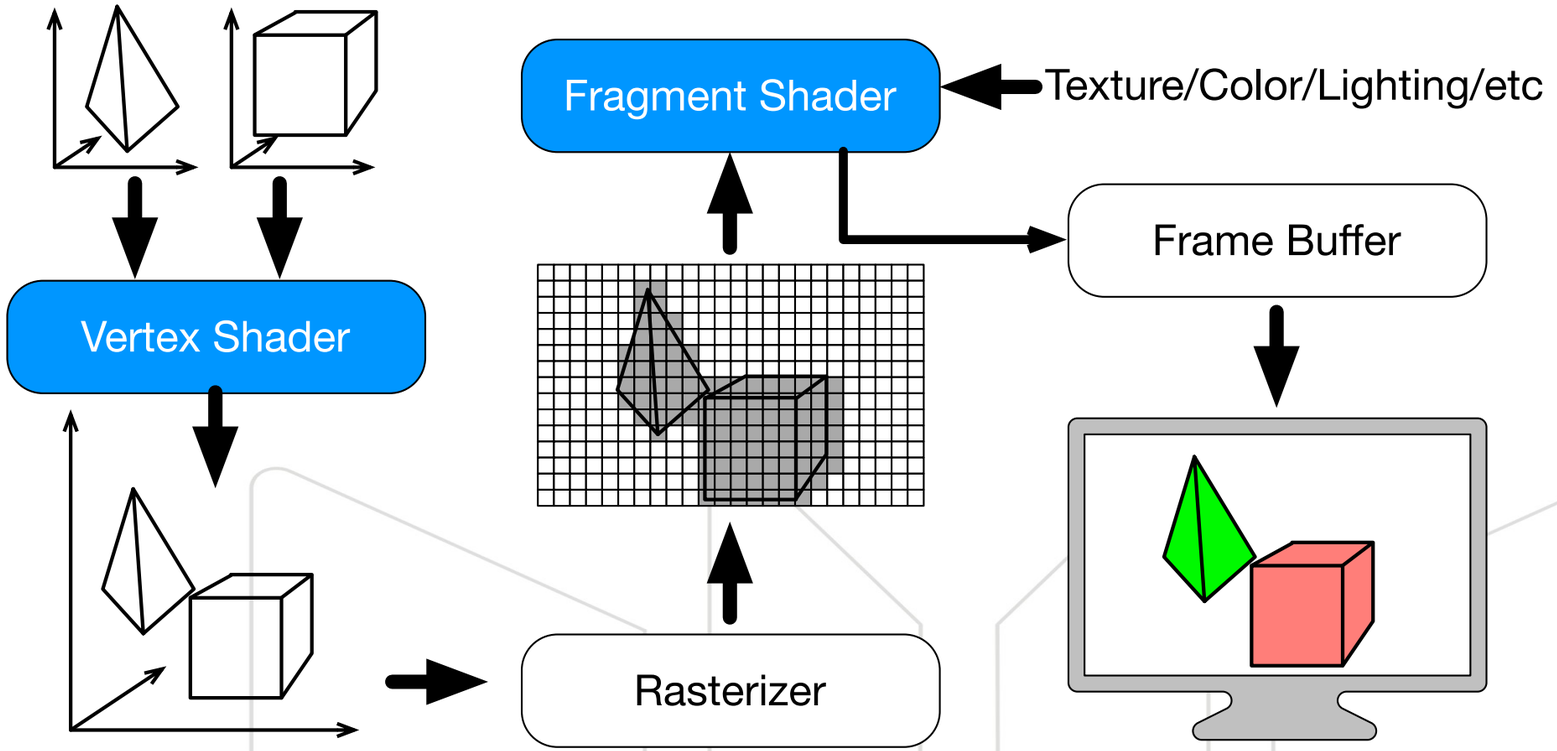
- OpenGL（GL：graphics library）
  - 低级别图形编程API
  - 超过250个函数
  - 由Silicon Graphics Inc.（SGI）在1992年开发
  - 最新版本4.6，发布于2017年7月31日

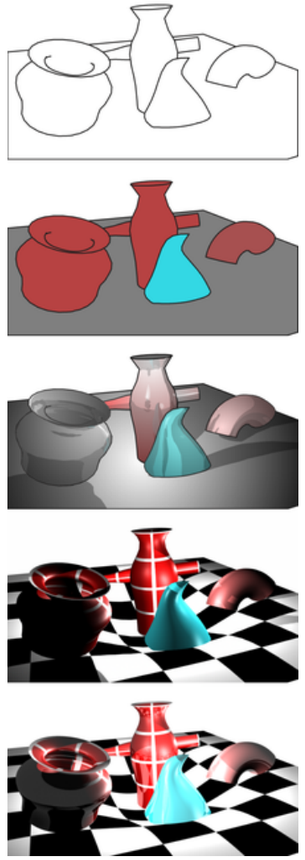




OpenGL简介

- OpenGL用途：渲染
  - 以软件由模型生成图像的过程
    - 语言或数据结构进行严格定义的三维物体或虚拟场景的描述
    - 几何体、视点、纹理、照明等

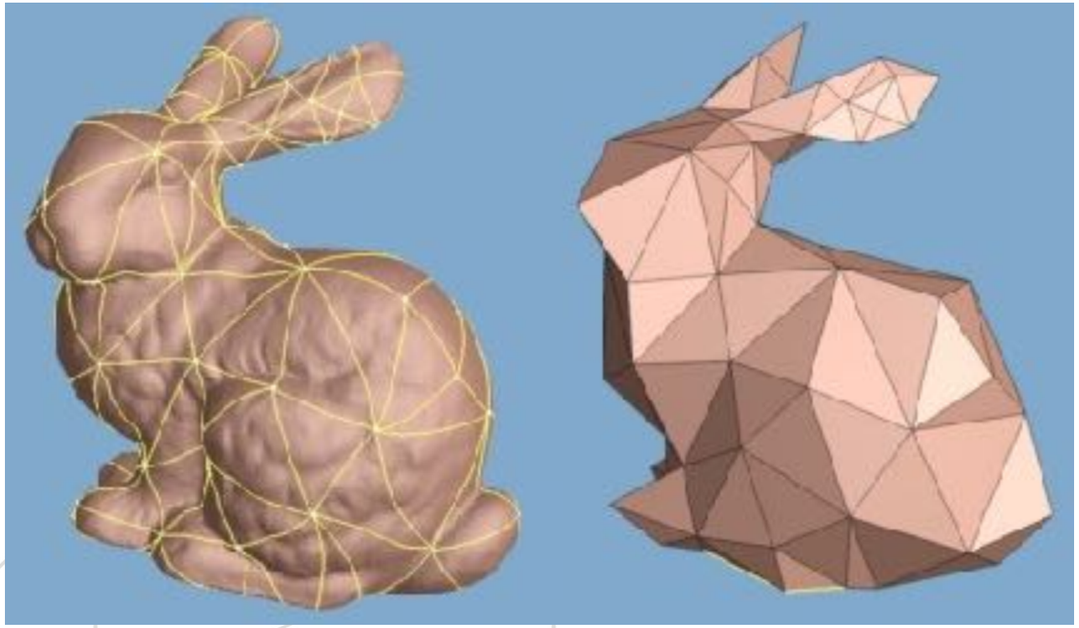




图片来自Wikipedia

OpenGL简介

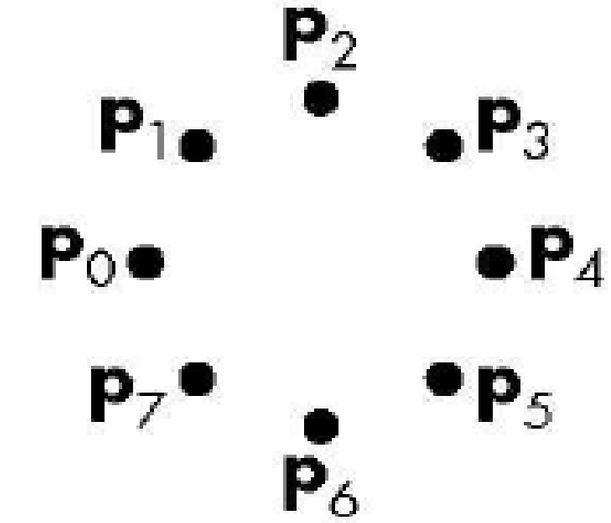
- OpenGL基本语句
  - 几何模型常表示为一组primitives
  - OpenGL中绘制primitives的基本语句
    - `glColor4f(r, g, b, a);`
    - `glBegin(primitive_type);`
    - `glVertex3f(x,y,z);`
    - `glEnd();`
  - 视角及物体的平移旋转由矩阵表示
    - `glOtho()`
    - `gluPerspective()`
    - `glTranslate()`
    - `glRotate()`



OpenGL简介

- 使用OpenGL绘制primitives
  - 举例：绘制GL_POINTS绘制点云
    - 点的大小可由glPointSize (size)指定

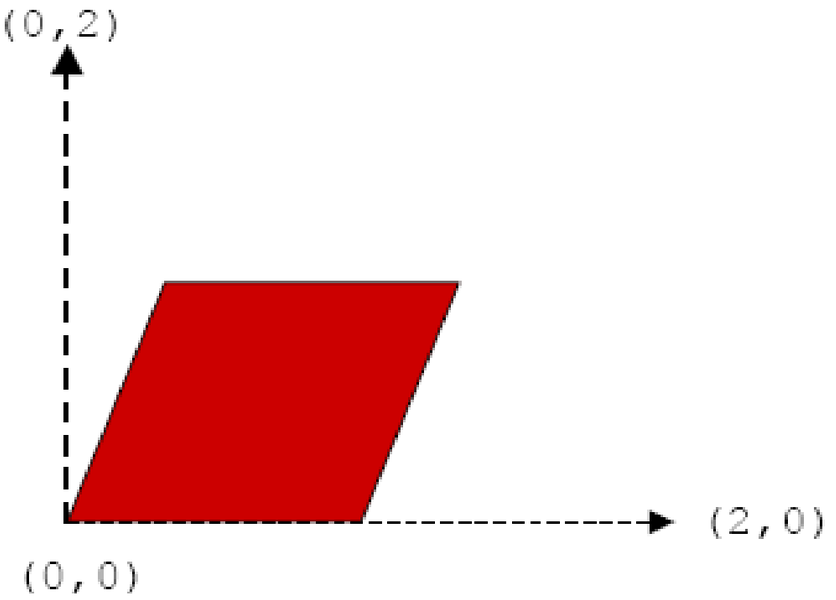
```
glBegin(GL_POINTS);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



OpenGL简介

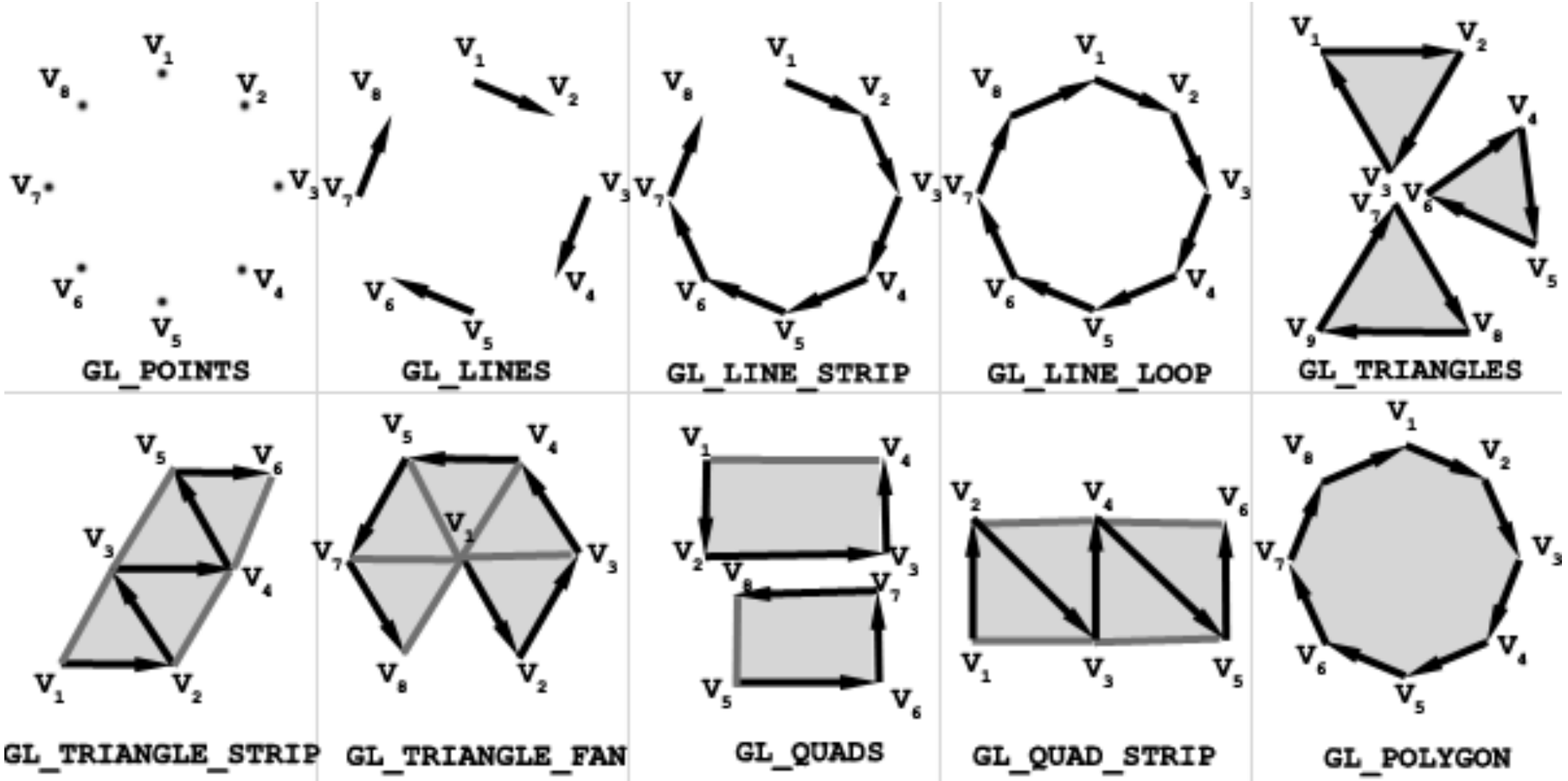
- 使用OpenGL绘制primitives
  - 举例：绘制GL_QUADS绘制平行四边形

```
void drawParallelogram( GLfloat color[] ){
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```



OpenGL简介

- 使用OpenGL绘制primitives
  - OpenGL支持的primitives



图片来自www.3dgep.com

课程提纲

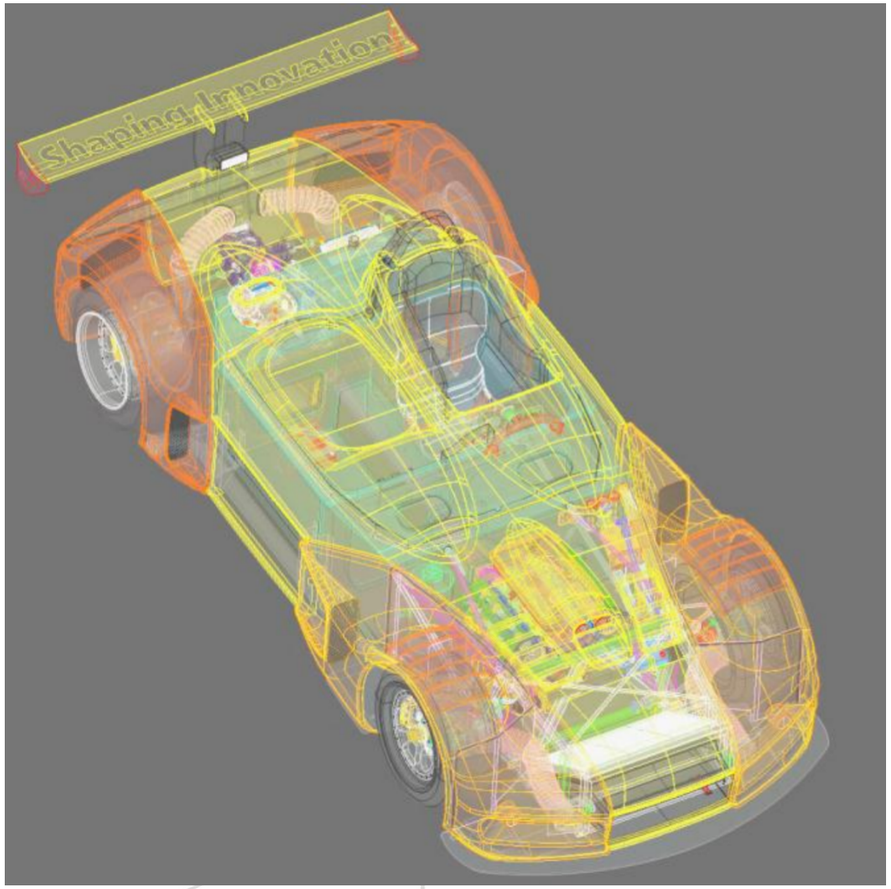
- OpenGL简介
- CUDA与OpenGL交互
- 应用举例

CUDA与OpenGL交互

- 使用OpenGL绘制primitives
  - 基本函数调用结构

```
glBegin(...);
for (int i=0; i<n; ++i){
    glColor(...);
    glVertex(...);
}
glEnd();
```

  - 每次绘制都将重复调用glColor, glVertex数百万次，并将数据从主内存传至显卡！



汽车模型：3,700,000个三角形

图片来自NVIDIA

CUDA与OpenGL交互

- 使用OpenGL绘制primitives
  - 解决方案：创建VBO（vertex buffer object），将模型一次传输到显存中，并使用一次CPU调用进行绘制

```
glBegin(...);
for (int i=0; i<n; ++i){
    glColor(...);
    glVertex(...);
}
glEnd();
```

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);

glVertexPointer(4, GL_FLOAT, 0, 0);
glColorPointer(4, GL_FLOAT, 0, (void*)colors_offset);

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);

glDrawElements(GL_QUADS, num_quads, GL_UNSIGNED_INT, 0);
```

CUDA与OpenGL交互

- 创建CUDA能访问的VBO
  - 使用**glGenBuffers**创建VBO， **glBindBuffer**为VBO分配空间
  - 使用**cudaGraphicsGLRegisterBuffer**将VBO注册为CUDA资源

```
void createVBO(GLuint* vbo, const unsigned int &size,
               struct cudaGraphicsResource **vbo_res, unsigned int vbo_res_flags)
{
    *vbo = 0;
    // create buffer object
    glGenBuffers(1, vbo);
    glBindBuffer(GL_ARRAY_BUFFER, *vbo);

    // initialize buffer object
    glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    // register this buffer object with CUDA
    cudaGraphicsGLRegisterBuffer(vbo_res, *vbo, vbo_res_flags);
}
```

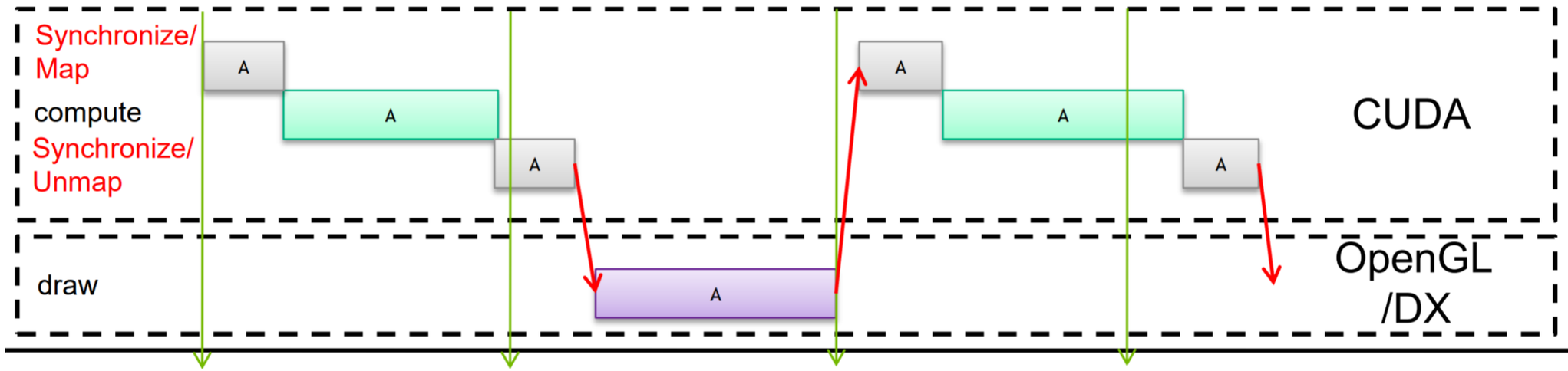
CUDA与OpenGL交互

- 通过CUDA对VBO中模型进行计算/修改
  - 使用前调用**cudaGraphicsMapResources**
  - 使用后调用**cudaGraphicsUnmapResources**

```
cudaGraphicsMapResources(1, &vbo_res, 0);
cudaGraphicsResourceGetMappedPointer((void **)&vertices_d, &num_bytes, vbo_res);

do_something_with_vertices_on_gpu<<<...>>>(vertices_d);

cudaGraphicsUnmapResources(1, &vbo_res, 0);
```



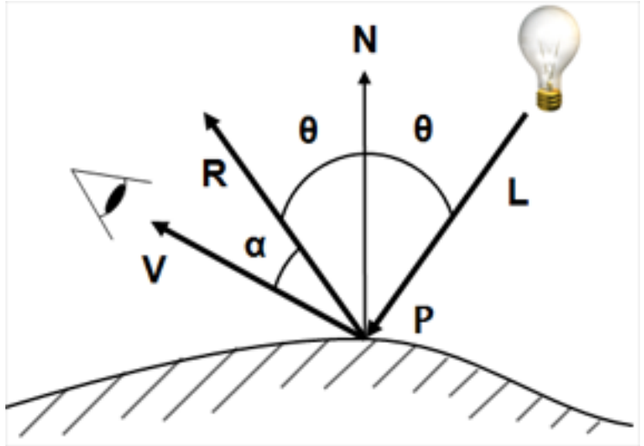
图片来自NVIDIA

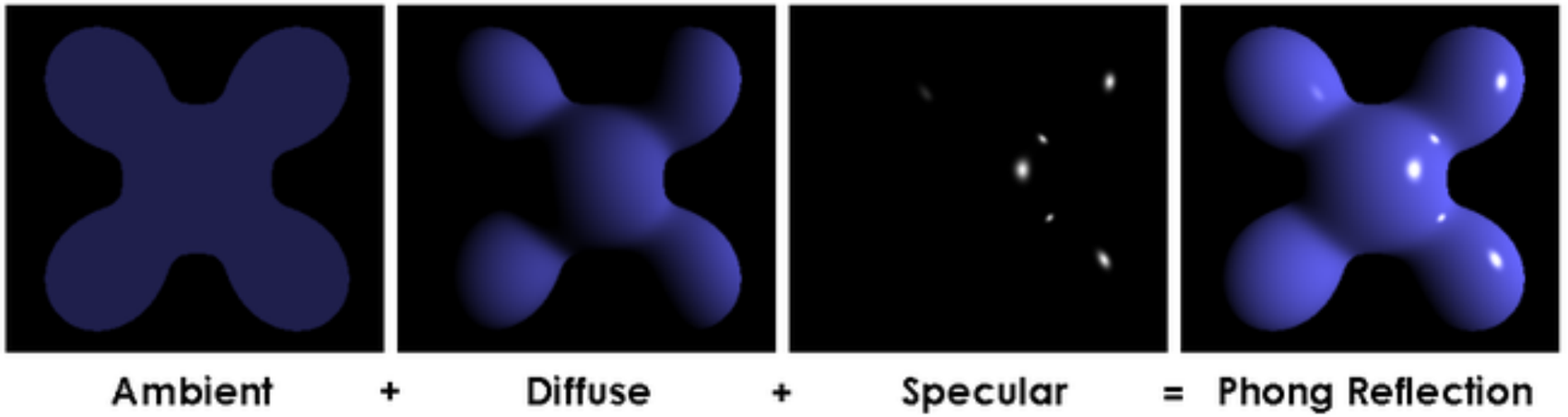
课程提纲

- OpenGL简介
- CUDA与OpenGL交互
- 应用举例

CUDA与OpenGL交互举例

- 光照模型（illumination）
  - OpenGL常用的Phong Shading是local illumination
    - 由美籍越南裔学者裴祥风提出（Bui Tuong Phong）
    - 对每个像素都需要计算
      - Ambient环境反射光
      - Diffuse漫反射光
      - Specular镜面反射光

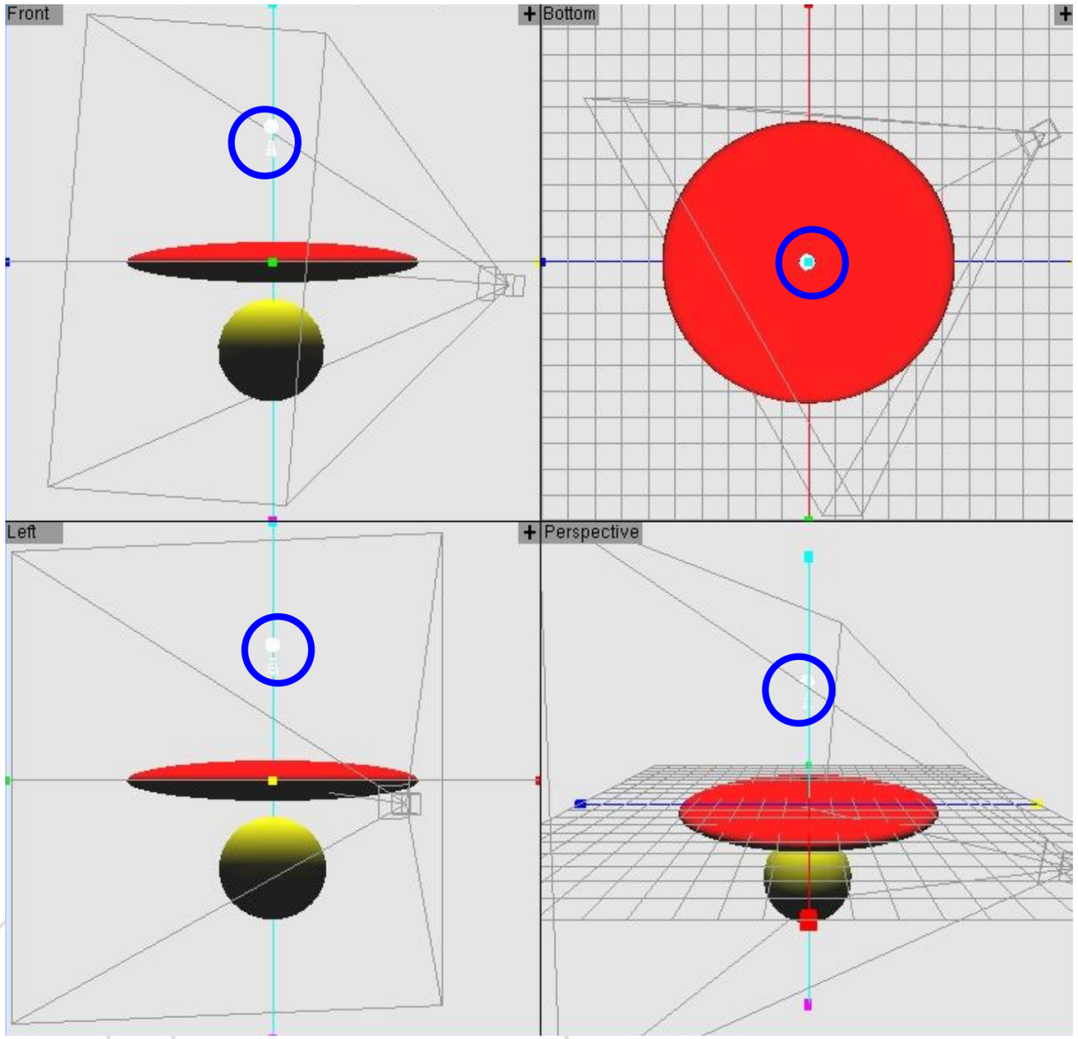




图片来自Wikipedia

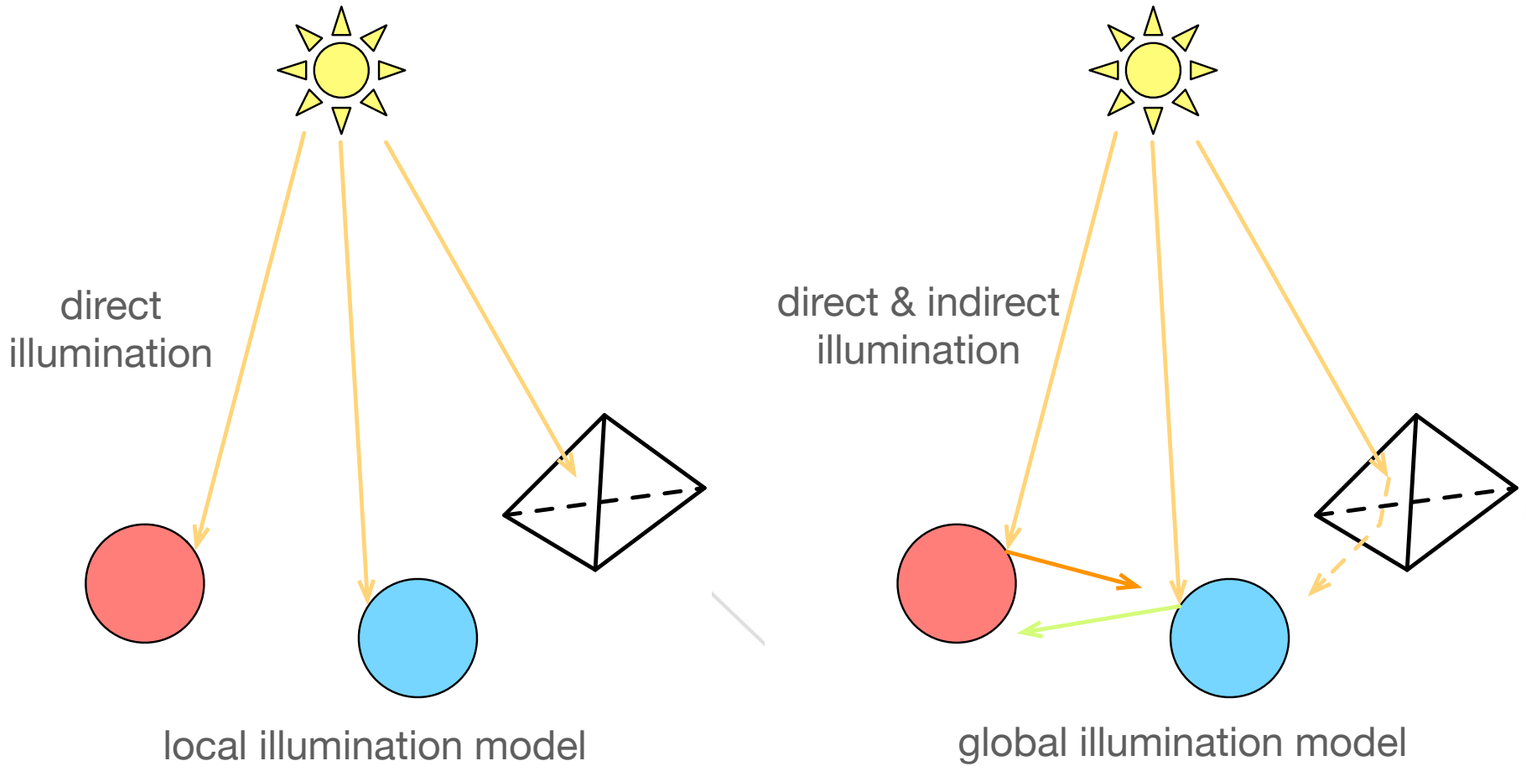
- 光照模型（illumination）
  - Local illumintation的问题
    - 每个点的颜色取决于
      - 光源、视点、该点处法向量
      - 物体与物体之间、物体不同部分之间没有关系！

问题：多个物体或物体不同部分与光源之间位置关系对光照结果没有影响！



图片来自Ching-Kuang Shene课件

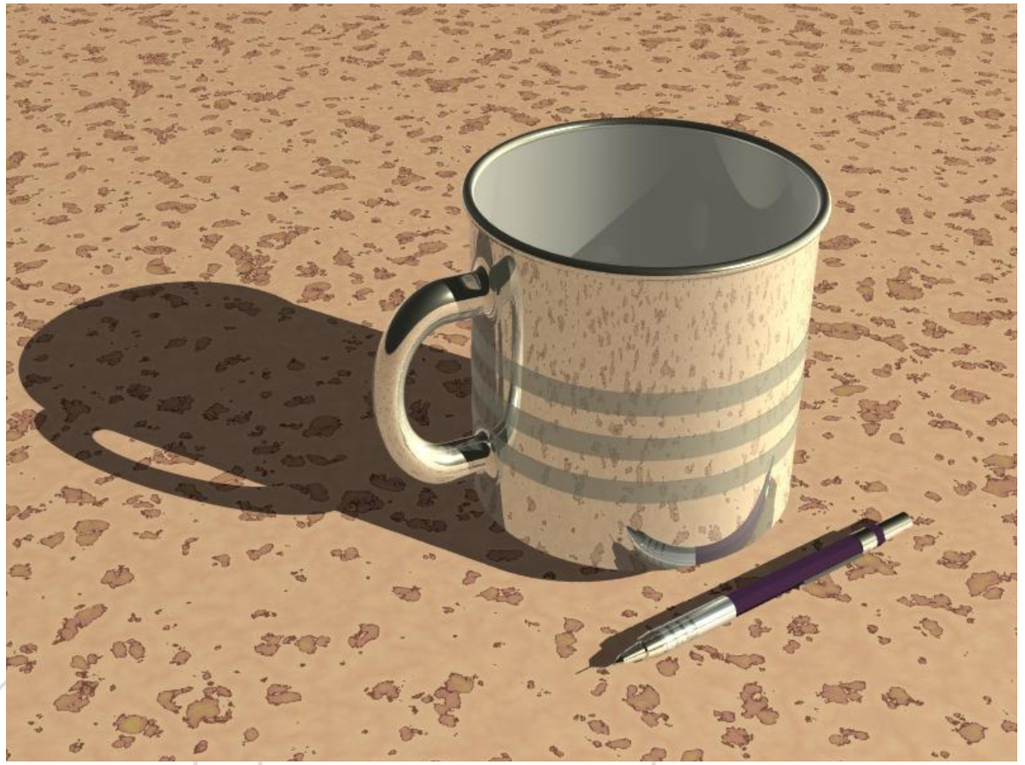
- 光照模型（illumination）
  - Local与global illumination
    - 在local模型中，每个点的颜色由局部信息决定
    - Global模型中，需要考虑物体与光之间的关系



CUDA与OpenGL交互举例

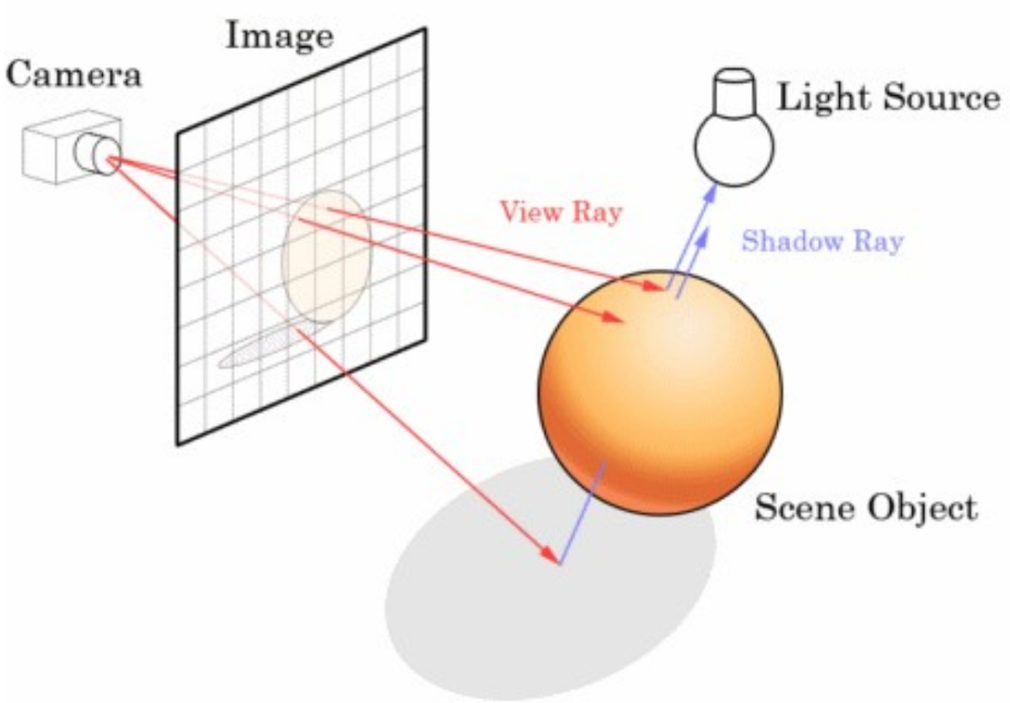
- Ray tracing结果举例
  - 图片产生于开源图形学渲染引擎POV-Ray
    - www.povray.org

图片来自wikipedia，使用POV-Ray 3.6生成。



2009年，我使用POV-Ray 2.7生成的图像。

- Global Illumination
  - 基本思路：
    - 从视角出发追踪光线（ray tracing）
    - 从光源出发追踪光线（photon mapping）

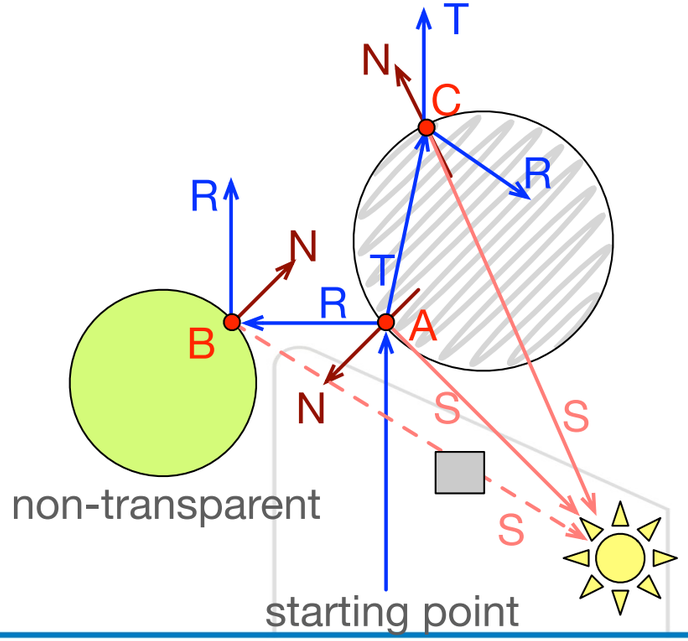


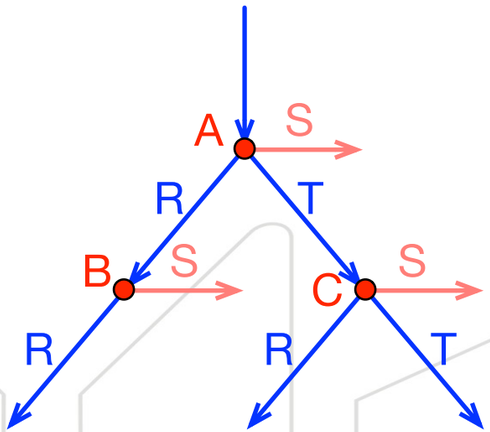
CUDA与OpenGL交互举例

Global Illumination

– Ray tracing

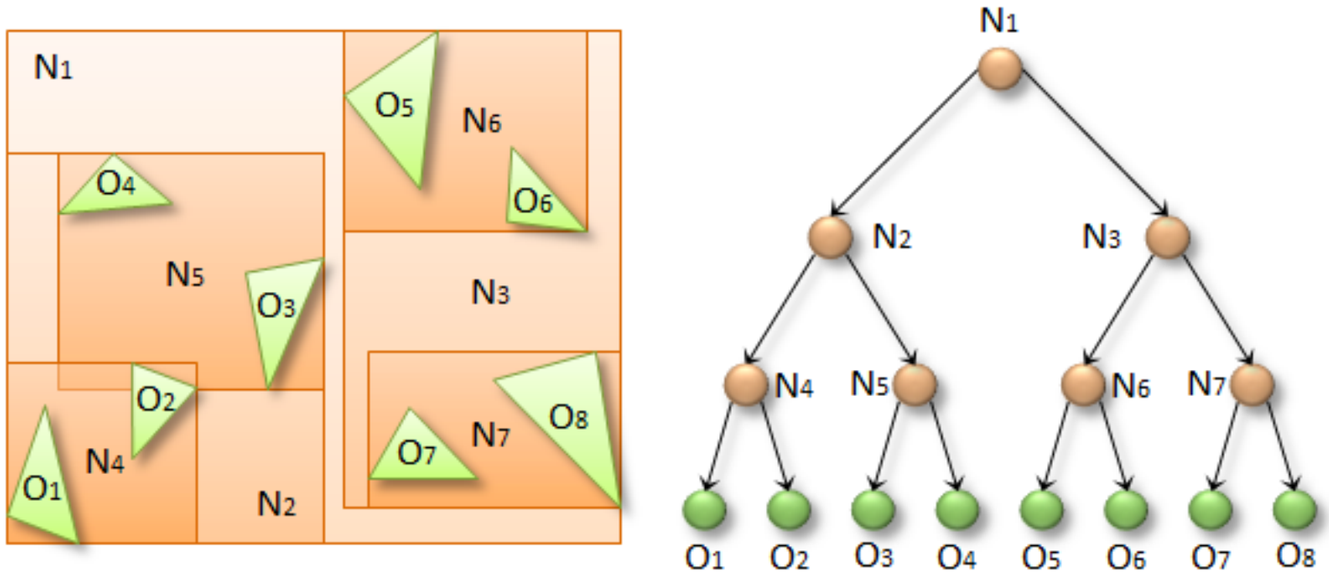
- 初始化时产生与图像像素相同数目的光线数目进行追踪
  - 当光线打在物体上时，产生三道光线进行进一步追踪
    - » 反射光R，折射光T，以及阴影光S
- 光线追踪过程的关键为直线与几何物体的碰撞检测





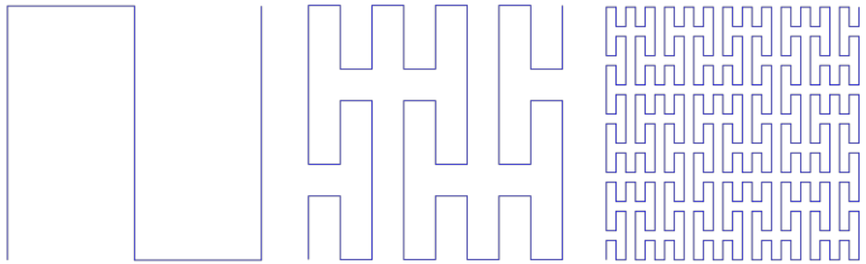
CUDA与OpenGL交互举例

- 使用CUDA进行碰撞检测
  - 所有几何体数据都已经在VBO中，无需拷贝
  - bounding volume hierarchy (BVH)-tree
    - 使用包围盒（bounding volume）将物体分配到树的节点中
    - 在查找最近邻时，先判断与包围盒的距离

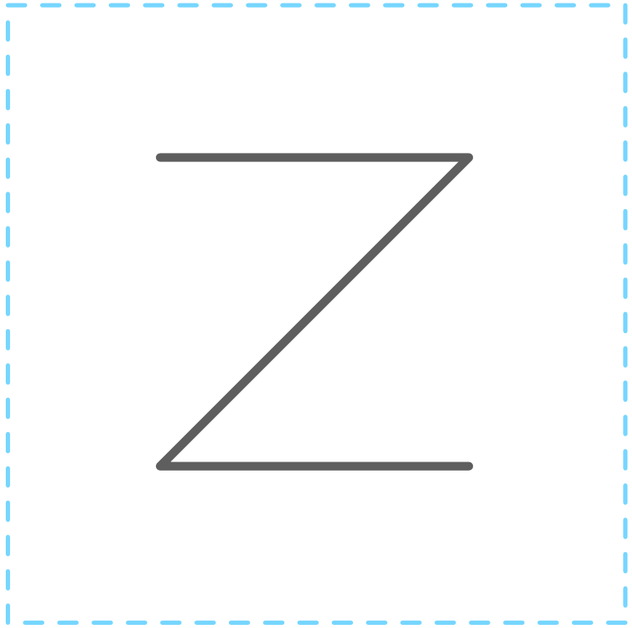


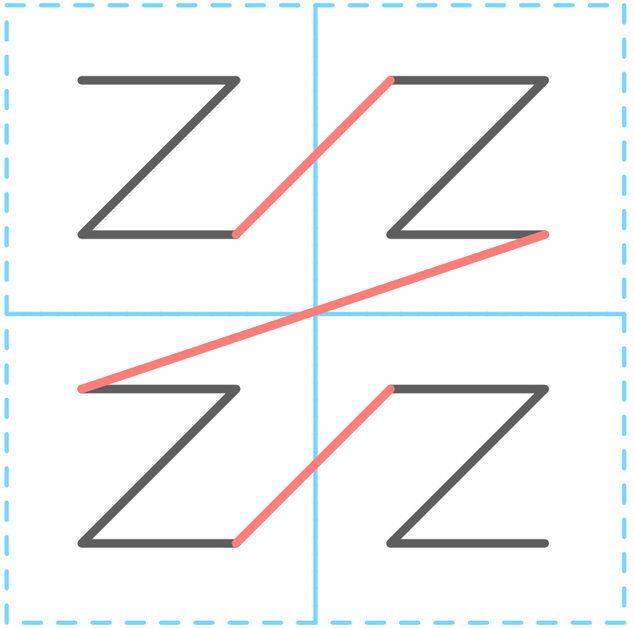
图片来自 https://devblogs.nvidia.com/thinking-parallel-part-ii-tree-traversal-gpu/

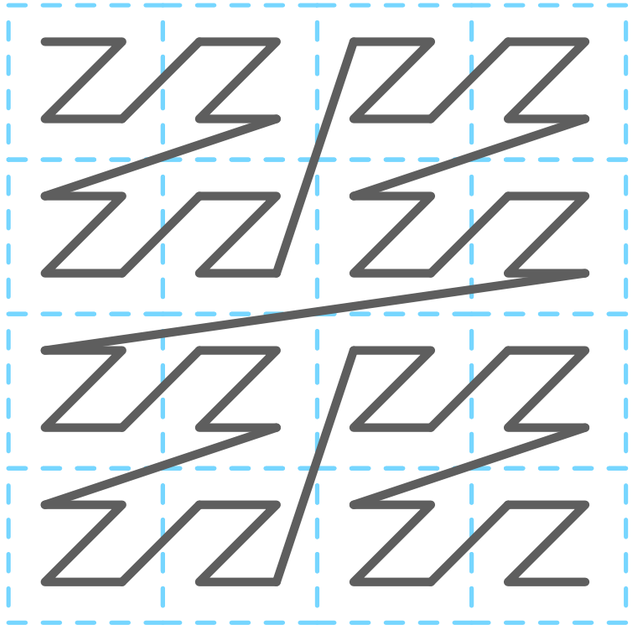
- 构建BVH-tree（Karra’s algorithm)
  - 1. 将所有几何体沿空间填充曲线排列
    - 沿 Z-order curve 排列
    - 使用一维曲线填充高维空间
    - 源于分形几何



Peano Curve. 图片来自Wikipedia.

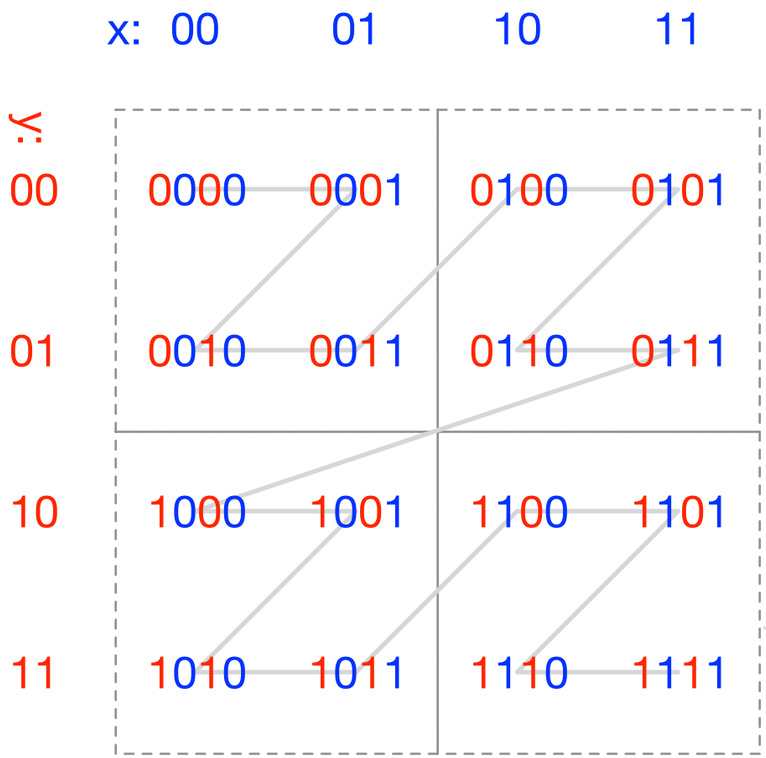


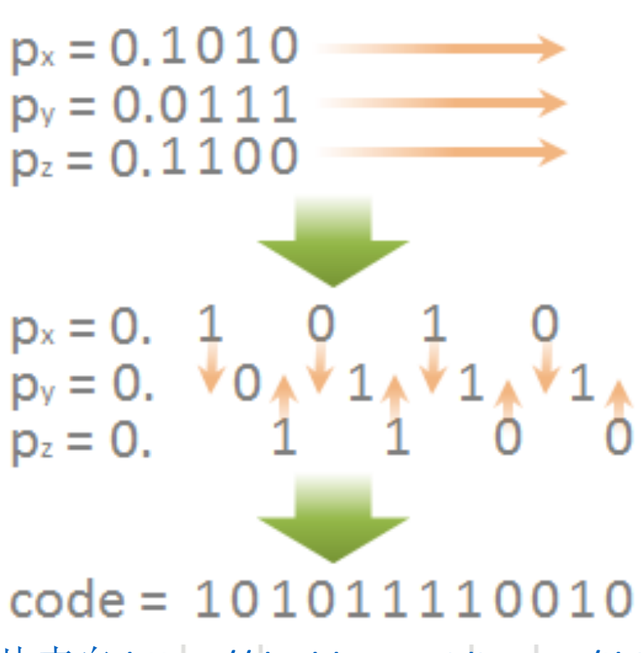


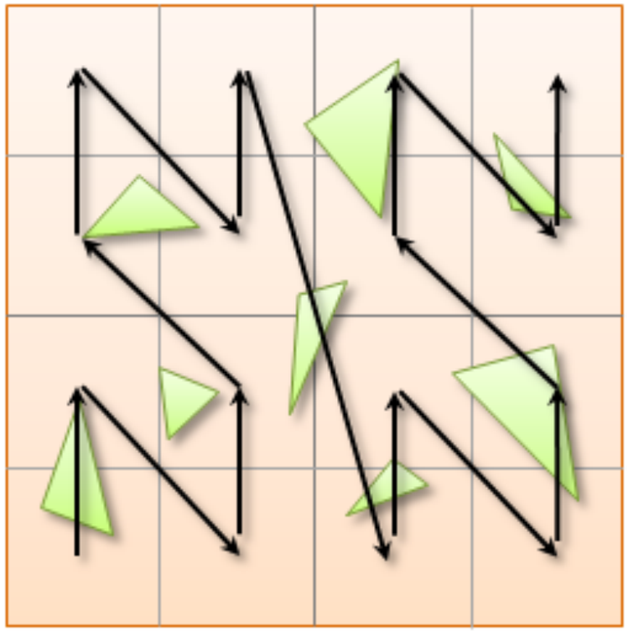


CUDA与OpenGL交互举例

- 构建BVH-tree（Karra’s algorithm)
  - 1. 将所有几何体沿 Z-order curve 排列
    - 将三维坐标转为Morton codes
    - 将几何体按其Morton codes排序 （radix sort）

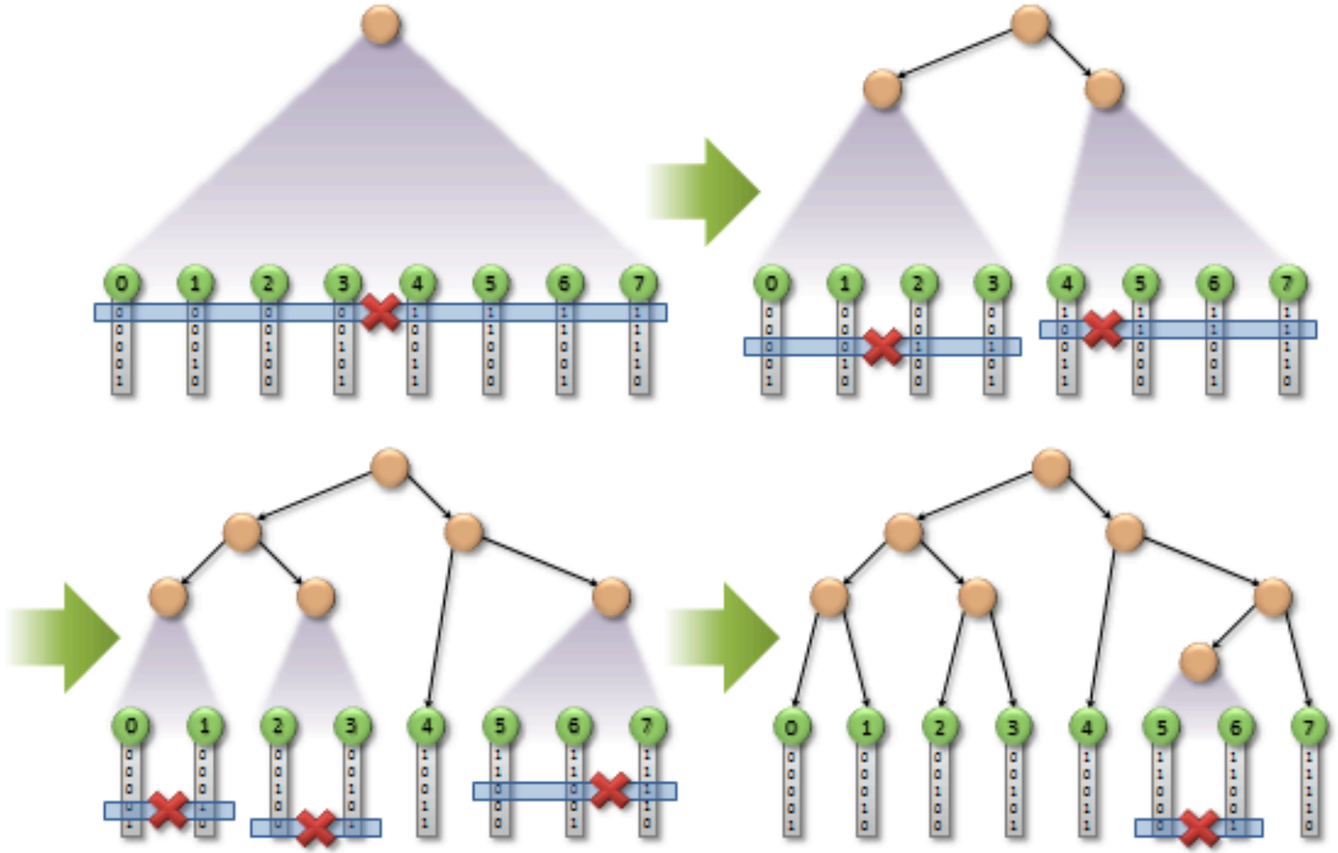






图片来自 https://devblogs.nvidia.com/thinking-parallel-part-iii-tree-traversal-gpu/

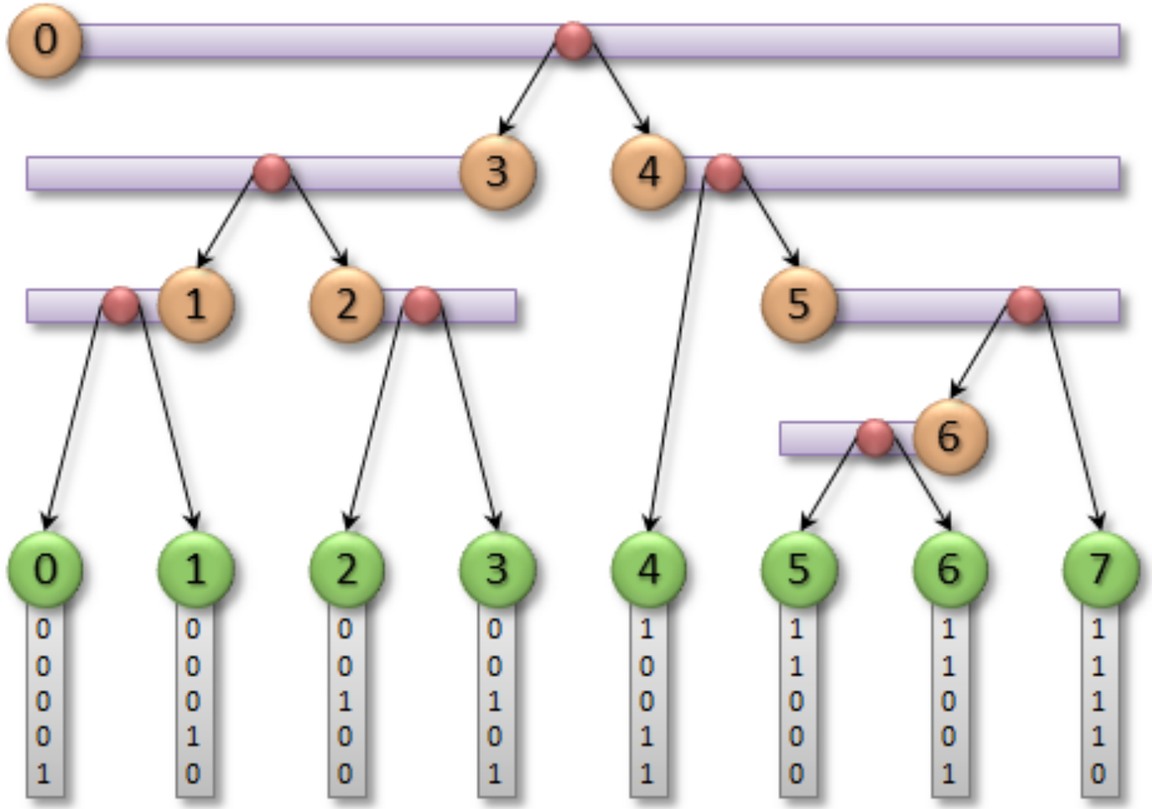
- 构建BVH-tree（Karra’s algorithm)
  - 2. 将排列好的几何体分段
    - 根据Morton codes最高不同位分段（可并行）



存在问题？

图片来自 https://devblogs.nvidia.com/thinking-parallel-part-iii-tree-traversal-gpu/

- 构建BVH-tree（Karra’s algorithm)
  - 2. 将排列好的几何体分段
    - 将几何体放入相应的range中



图片来自 https://devblogs.nvidia.com/thinking-parallel-part-iii-tree-traversal-gpu/

Questions?