**光线追踪实验报告**

计24 黄晨 2012011337

目录

[1 程序描述 3](#_Toc392516948)

[2 程序类介绍 3](#_Toc392516949)

[2.1 Camera类 3](#_Toc392516950)

[2.2 Scene类 3](#_Toc392516951)

[2.2.1 Object类 3](#_Toc392516952)

[2.2.2 Sphere类、Rectangle类、Triangle类及TriangleOBJ类 3](#_Toc392516953)

[2.2.3 Texture类 4](#_Toc392516954)

[2.2.4 RaySource类 4](#_Toc392516955)

[2.2.5 场景类 4](#_Toc392516956)

[2.2.6 部分static成员 4](#_Toc392516957)

[2.2.7 成员函数说明 4](#_Toc392516958)

[2.3 Raytrace类 5](#_Toc392516959)

[2.4 Movie类 5](#_Toc392516960)

[2.4.1 Movement类 5](#_Toc392516961)

[2.4.2 CameraMove类 5](#_Toc392516962)

[2.4.3 BackgroundMgr类 5](#_Toc392516963)

[3 程序算法描述 5](#_Toc392516964)

[3.1 基本的光线追踪 5](#_Toc392516965)

[3.2 抗锯齿 6](#_Toc392516966)

[3.2.1 FSAA 4x 6](#_Toc392516967)

[3.2.2 Smooth 6](#_Toc392516968)

[3.3 光线追踪加速 6](#_Toc392516969)

[3.3.1 八叉树加速 6](#_Toc392516970)

[3.3.2 多线程加速 6](#_Toc392516971)

[3.4 高维纹理 7](#_Toc392516972)

[3.5 动画 7](#_Toc392516973)

[4 程序亮点 8](#_Toc392516974)

[4.1 Obj的导入 8](#_Toc392516975)

[4.2 场景文件 8](#_Toc392516976)

[4.3 自定义动画 9](#_Toc392516977)

[4.4 加速 10](#_Toc392516978)

[4.5 高维纹理 10](#_Toc392516979)

[4.6 背景渐变 10](#_Toc392516980)

[4.7 键盘控制 10](#_Toc392516981)

[5 程序运行结果 10](#_Toc392516982)

[5.1 普通光线追踪与两种抗锯齿效果 10](#_Toc392516983)

[5.1.1 普通光线追踪 10](#_Toc392516984)

[5.1.2 FSAA4x抗锯齿 11](#_Toc392516985)

[5.1.3 Smooth平滑 11](#_Toc392516986)

[5.2 普通光线追踪——Dragon 12](#_Toc392516987)

[5.3 多obj 12](#_Toc392516988)

[5.3.1 普通光线追踪 12](#_Toc392516989)

[5.3.2 FASS4x抗锯齿 13](#_Toc392516990)

[5.3.3 FASS4x抗锯齿+Smooth平滑 13](#_Toc392516991)

[5.4 动画 14](#_Toc392516992)

[6 总结与收获 14](#_Toc392516993)

[6.1 OpenCV 14](#_Toc392516994)

[6.2 遇到的困难 14](#_Toc392516995)

[7 参考资料 15](#_Toc392516996)

1. 程序描述

本程序实现了基本的光线追踪，使用八叉树进行了加速，另外支持自定义场景，自定义纹理，自定义动画，以及obj导入。另外在图片渲染模式下，一幅图像渲染完成后，可通过键盘进行上下左右的移动以及视角方向的改变（如平移、旋转、俯仰等），当然，由于性能限制，每次移动根据场景复杂度需要等待几秒至几分钟不等。

1. 程序类介绍
   1. Camera类
      1. Ray类：用以定义一根光线，由出发点和方向表示，其中方向向量应被单位化。
      2. 照相机类：一个照相机由屏幕像素的宽度和高度、屏幕实际的大小（即坐标系内的大小）、视点坐标、朝向以及正上方方向确定。
      3. 获取光线列表：这里有两个函数均可获取光线列表，一个是getRay()，另一个是getRay\_antiAliasing()，前者返回一个vector容器，其中是需要追踪的所有光线，后者返回一个vector的数组（共4个vector），分别为每个像素分解后的光线（此处在3程序算法描述中的抗锯齿部分将进行说明）。
      4. 运动：Camera类内置了move、rotate等用于相机运动的函数，另外也可通过set函数直接设置相机的位置。在动画生成是通常使用set，在键盘控制的情况下使用内置的移动函数。
   2. Scene类

在介绍此类之前，先对其使用的一些类做一些说明。

* + 1. Object类
       1. 物件类：用于定义各个部件的抽象类，其中包含了每个物件的材质信息（颜色color、漫反射率diffuse、镜面反射率specular、透射率transmiss、环境光反射率environmentR、折射率refractivity和高光指数s）、物件名称以及获得物件各信息的接口函数。
       2. CrossRay(…)函数：有两个版本，其一返回值为bool，即判断是否相交，并利用传入的参数返回交点坐标和相交类型；另一个返回值为int，该函数只判断给定线段是否与该物件相交，返回相交次数。
          1. bool型函数，当返回值为真时，光线段与该物体相交，P为交点坐标，inside为相交类型。其中，相交类型分为3种，0型、1型和2型。

0型：光线从外部进入物件内部。

1型：光线从内部穿出至物件外部。

2型：光线从物体外部射出，与物体相交后仍在物体外部。该类型主要用于面片以及不用区分内外的物件。

* + - * 1. Int型函数，直接返回相交次数。返回值为0表示不相交。
      1. InScene函数：输入为场景的前左下角坐标和后右上角坐标，返回值为bool型。返回真表示该物体与场景有交，否则返回假。
      2. 运动：每个物件也相应定义有运动相关的函数，每个函数返回值为bool类型，表示是否可以进行对应的运动及是否成功完成运动，默认返回值为假，即不可运动。
      3. involvedScene：set容器，用以存储与该物件有交的所有场景，具体作用在3程序算法描述中的动画渲染加速部分将做进一步说明。
    1. Sphere类、Rectangle类、Triangle类及TriangleOBJ类
       1. 均继承自Object类，分别对应为球、平行四边形、三角形以及区分正反面的三角形。最后的TriangleOBJ主要为Obj设计，在将Obj作为可透光物件时使用。
       2. 根据物件不同，每个类也定义有自己专用的构造函数。
    2. Texture类
       1. 该类对纹理图像进行了封装，将图像转化为颜色数组方便以后的使用。
       2. 通过Load函数读取纹理图，通过isLoaded检测是否成功载入。
       3. 重载了[]操作符方便使用。
    3. RaySource类

光源类，定义了每个点光源的位置和颜色。

* + 1. 场景类

由前左下角坐标和后右上角坐标定义一个标准场景，由Background指定背景纹理标号（默认将背景映射为球型纹理）。minDistance制定了最小场景边长限制，maxObjects指定了最大物件个数限制。son为子场景列表，father为父场景，id为场景在其父场景的子场景中的标号。初始场景的father为0。

* + 1. 部分static成员
       1. RaySources：所有光源的列表
       2. \_\_LFD、\_\_RBU：初始场景的前左下角坐标和后右上角坐标
       3. AllObjects：所有物件的列表
       4. WaitObjects：等待重新分配场景关系的物件的列表
       5. nextBackground、alpha：用于动画中的背景变换使用，在在3程序算法描述中的动画部分将做进一步说明。
       6. textures：所有纹理的列表
       7. radius, radius\_sqr：球型纹理映射时的半径大小，在初始化的时候将自动进行更新。
    2. 成员函数说明
       1. outPos(Ray&)：计算光线在不与物体相交的情况下射出场景时的坐标。
       2. cross(…)：同样为两个版本，一个版本用于求交，一个版本用于计算阴影。
          1. bool cross(Ray&, SimpleOBJ::Vec3f&, Object \*&, int &)：求交用。当返回值为真时，返回相交物件的指针和相交类型（定义见2.2.1.2 1），若相交物件指针为0，则表示已射出最大场景；当返回值为假时，返回射出该场景时坐标。
          2. bool cross(Ray& ray, SimpleOBJ::Vec3f& rs\_pos, std::vector<Object\*> \*objs, bool &flag, SimpleOBJ::Vec3f &pos)：计算阴影用。当返回值为真时，objs为在该场景中与光线相交的物件列表，若objs为空，则已到底最大场景的边缘。另外，在返回值为真时，若flag为真，则到达光源，若flag为假，则pos为射出时的坐标；当返回值为假时，pos表示射出时的坐标。
       3. Initialize(bool)：初始化函数，输入的bool表示是否为第一次初始化，即是将AllObjects里的物件分配至各个场景，还是将WaitObjects中的物件进行再分配。关于分配物件将在3程序算法描述中的八叉树加速部分进行进一步说明。
       4. shink()：这个函数主要用于动画生成时，在物件发生移动之后，对于那些已经空了的场景，若其有子场景，则将子场景删除时使用。
  1. Raytrace类
     1. 光线跟踪类，其中定义了BRDF模型的计算方式、折射与反射函数、光线追踪接口以及平滑函数。一个光线跟踪类包含了一个相机、一个场景，同时定义了最大递归深度。
     2. 折射与反射：通过调用Object的方法获取物件某点的法相与材质，进而生成折射光与反射光进行进一步追踪。
        1. 关于折射：当光线从光密介质到达光疏介质时，在入射角较大的情况下可能发生全反射，因此折射函数的返回值为一个int，表示是否发生全反射，以及折射前后光线与物件内外的关系，用于计算折射率进行下一层递归。当发生全反射时，镜面反射虑变为原镜面反射率加上透射率。
     3. inScene函数：根据光线的起点与方向判别该光线是否在某场景内。
     4. subScene函数：获取包含某光线的最小场景。
     5. Trace函数：光线跟踪主函数，完成光线跟踪的各个阶段，返回颜色列表。
     6. trace函数：光线跟踪函数，输入一根光线与起始场景，返回颜色值。
     7. rsTrace函数：计算阴影时使用的追踪函数。
  2. Movie类

用于生成动画的类，包含Movement类、CameraMove类和BackgroundMgr类。

* + 1. Movement类
       1. 控制物件运动的类，定义了一个运动信息类MoveInfo，Movement类在每次移动物件时根据MoveInfo的信息做出相应调整。
       2. movements成员：一个运动序列的vector容器，内部元素依照起始帧与结束帧的大小进行了排序。
       3. Init()：初始化，将当前帧置0。
       4. move()函数：根据movements和当前帧序号对物件进行移动。
    2. CameraMove类
       1. 控制相机运动的类，由视点、前点与顶点确定相机的位置信息。
          1. 前点：相机正前方向上的一点
          2. 顶点：确定相机正上方的点，不一定恰好位于正上方，其与视点、顶点构成的平面即觉得了相机的正上方。这样设计是便于对相机的朝向进行设置。
       2. 为了方便起见，将视点、前点与顶点均设置为了一个球型物件（Sphere），只不过这三个球体并未加入值场景类中的AllObjects中，因此不会被显示。CameraMove类内含三个Movement类对这三个球的运动进行控制。
    3. BackgroundMgr类
       1. 控制场景背景变换的类，与该类配套的同样定义了一个BackgroundInfo类来存储背景变换信息。
       2. 与Movement类的实现方式类似，backgoundList存储了背景变换的各种信息，然后BackgroundMgr类中存储了当前的帧数。通过init()进行初始化，通过next()进行场景变换。

1. 程序算法描述
   1. 基本的光线追踪
      1. 对每一跟光线，调用场景类的cross方法求出光线与场景中物体的交点，调用光线追踪类的refract和reflect方法生成折射与反射光，对折射与反射光进行继续追踪，之后利用着色函数进行着色，返回颜色值。
      2. 追踪停止的条件：超过最大递归深度或光线对色彩的贡献过小。
   2. 抗锯齿
      1. FSAA 4x

本程序使用的抗锯齿算法为FSAA全屏抗锯齿，采样虑为4倍，即将每一个像素点分为4各小的像素点分别采样，最后的结果用4各像素点的颜色取平均值。

* + 1. Smooth

后期平滑处理。即将每个点的颜色换位其本身、右方、下发、右下方4个点颜色的平均值，这种处理方式可以达到一定的抗锯齿效果，不过会使画面整体模糊化。虽然性能不必FSAA，不过速度较快。

* 1. 光线追踪加速
     1. 八叉树加速
        1. 初始化：
           1. 若场景尺度小于最小尺度，或场景中物件个数小于最大物件个数，返回。
           2. 创建子场景，对每个物件，检测其与所有子场景是否相交，若相交则将其加入对应子场景的物件列表。
           3. 初始化各子场景，返回。
        2. 对每一个光线，首先求出包含其的最小场景，然后获取该光线与场景的交点坐标，这样，有光线起点和光线与场景的交点就构成了一跟线段。之后对场景中的每个物件，看该线段是否与其相交，若相交，求出最近交点，将最近交点作为起点，生成折射与反射光，继续进行追踪。若不相交，则将光线与场景的交点作为新的起点，继续进行追踪，直至相交或到达最大场景边缘。
        3. 关于获取包含光线的最小场景
           1. 根据课件上的场景标号原则，将光线起点与场景分割的中点比较即可。初始子场景标号置为0。若x坐标大于中点x坐标，标号加1，若y坐标大于中点坐标，标号加2，若z坐标大于中点z坐标，标号加4。最后所得即为子场景的标号。值得注意的是对临界情况的处理，若起点位于某一分割平面上，应根据其方向进行界定属于哪一侧。
           2. 初始状态的查找：对每一个场景，若其无子场景，返回该场景；否则根据1）的规则查找起点所属的场景标号，进入对应的子场景，再次查找。
           3. 射出场景时的查找：在父场景中进行检测，若新的光线属于父场景，则在父场景中，使用1）的方法获得最小子场景；否则，将当前场景置为其父场景，重新对当前场景的父场景进行包含检测。
     2. 多线程加速
        1. 初始化多线程加速
           1. 主线程

若初始场景尺度小于最小尺度，或AllObjects中物件个数小于最大物件个数，则将AllObjects中所有物件加入初始场景的物件列表，将该场景指针加入各物件的involvedScene中，返回。

初始化各子场景，其中中点坐标使用各物件中心坐标的平均值。若以此法求出的中点坐标距离场景边界的某平面的距离小于1，则将中点坐标置为场景正中心。

对AllObjects中的每个物件，检测其与各子场景是否相交，若相交则将其加入对应子场景的物件列表。

对每个子场景，开启一个从线程进行初始化处理。

* + - * 1. 从线程

若场景尺度小于最小尺度，或场景中物件个数小于最大物件个数，将该场景指针加入各物件的involvedScene中，返回。

初始化各子场景，其中中点坐标使用各物件中心坐标的平均值。若以此法求出的中点坐标距离场景边界的某平面的距离小于1，则将中点坐标置为场景正中心。

对每个物件，检测其与所有子场景是否相交，若相交则将其加入对应子场景的物件列表，当该子场景的物件数大于最大物件数时，初始化该场景（这里进行初始化是为减少内存占用，并不节省时间）。

初始化各子场景，返回。

注：在多线程中维护各物件的involvedScene列表时，为避免访问冲突，应加入一个互斥锁。

* + - 1. 光线追踪的多线程加速
         1. 主线程将光线列表等分为4部分，开启4各从线程分别对4各光线列表进行光线追踪。
         2. 注意，由于分割为单纯的顺序分割，因此该加速在无抗锯齿的情况下加速效果为2-4倍，在抗锯齿的情况下加速效果为4倍。
  1. 高维纹理
     1. 场景背景采用的是球型纹理，对于每一个需要获取背景的背景点，根据其光线的前进方向，利用光线与球相交的算法，计算出其在背景纹理球上的交点，此处球心坐标取为场景最低（z坐标最小）平面的中心。
     2. 根据交点z坐标绝对值，使用映射函数，其中r为球半径（场景类中的radius），将其映射至纹理图的高度值（乘以最大高度）。
     3. 根据交点在最低平面的投影与y轴正半轴的夹角确定宽度值，映射函数，将其映射至纹理图的宽度值（乘以最大宽度）。
  2. 动画
     1. 基本运动函数：
        1. Move：运动至某点，或按某方向运动一定距离
        2. Rotate：根据跟定的点和轴方向进行旋转。旋转的函数在MyMath.h中进行了声明，在Object类中根据不同物体进行了封装。
     2. 每次进行运动时，根据当前帧数找出需要进行的所以运动。由于实现对运动列表进行了排序，因此可以顺序查找，当某个运动信息的起始帧大于当前帧时，即可退出查找。每帧对每个物体可能同时有多种移动。
     3. 对于每个运动，首先根据剩下的帧数和需要运动的距离或角度，计算出当前帧的移动参数，然后对对应物体进行移动。
     4. 背景变换
        1. 背景变换的原理
           1. 根据alpha值的不同，背景颜色也不同，当alpha实现渐变时，背景便实现了渐变。
        2. 根据变换的持续时间，计算出每一针alpha值的变化量。变换结束时更新背景。
  3. 动画渲染加速
     1. 动画渲染的原理就是一帧一帧地渲染然后将结果图像连接起来，但如果每一帧都删除之前的场景后重新初始化整个场景，将耗费大量的时间，因此本程序对动画渲染的初始化进行了优化。
     2. 动画渲染中的初始化加速：
        1. 对于每一帧，对每个移动过的物件，在包含该物件的场景中删除该物件，若删除之后场景不含有任何物件，则调用Scene类中的shrink方法对场景进行收缩（如可收缩的话）。将移动过的物件加入Scene类的WaitObjects中。
        2. 调用最大Scene中的初始化函数，传入参数置为false（非第一次初始化）。
     3. 依此法进行加速过后，每次只需对移动过的物件进行再分配，而不必进行整个场景的重构
     4. 缺陷：在很多帧以后，可能会存在一些冗余子场景，这是由于场景收缩的判断条件引起的。冗余子场景的数量和场景类中每个场景的最大物件数成正比。

1. 程序亮点
   1. Obj的导入
      1. 在八叉树的支持下，Obj的导入成为了现实。在不加速的情况下，渲染时间将非常的长。
      2. 导入Obj使用提供的parser即可，然后对每个三角面片，根据Obj的导入模式（在4.2中将进行介绍），初始化为一个Triangle或者TriangleOBJ即可。
   2. 场景文件
      1. 本程序的场景定义由文件进行，在运行程序时，可以通过参数输入场景文件名称，在不输入的情形下默认为scene.inf。
      2. 场景文件的格式
         1. #... 该行为注释
         2. window height width定义窗口大小
         3. scene LFD RBU定义场景大小，LFD为前左下角坐标，RBU为后右上角坐标
         4. smooth 0/1是否进行平滑：0表示不平滑，1表示平滑，不定义默认为不平滑
         5. antiAliase 0/1是否抗锯齿：0表示关闭抗锯齿，1表示开启抗锯齿，不定义默认为关闭
         6. back number背景的纹理编号
         7. texture path载入纹理
         8. camera sw sh focus pos forward upward定义相机信息，sw和sh为相机在坐标系中的实际宽度和高度，focus为焦距，pos为视点坐标，forward为正方向，upward为正上方，upward和forward可不垂直，不垂直时系统将自行修正
         9. depth number定义最大递归深度，不定义默认为2
         10. sphere center radius color diffuse specular transmiss environmentR refractivity s name定义一个球体，center为球心，radius为半径，其后的参数参见Object类的定义。
         11. triangle corner0 corner1 corner2 color diffuse specular transmiss environmentR refractivity s name定义一个三角面片，corner0/1/2分别为其三个顶点
         12. rect/rectangle leftDown length width color diffuse specular transmiss environmentR refractivity s name定义一个平行四边形，其中用/隔开的部分表示前后均可，leftDown为其左下角坐标，length和width均为向量，分别为长和宽。
         13. obj/object path name normal/obj/object center scale color diffuse specular transmiss environmentR refractivity s定义一个obj，其中normal表示普通模式，即不考虑obj的内外，obj/object为物体模式，此时光线在穿过obj时将发生折射（如果可能的话）。
         14. rs/raysource pos color定义一个点光源
         15. movie fileName一旦出现此语句，则输出为视频格式，其后的fileName指定了动画描述文件（在4.3中进行介绍），若不指定，则默认为Movie.inf
         16. save/savename fileName指定输出文件的文件名，默认为Image.bmp
         17. 注：
             1. 以上的color、diffuse、specular、transmiss、environmentR均为三通道，分别为r、g、b。
             2. 文件中坐标、向量均为x y z的格式
   3. 自定义动画
      1. 跟场景一样，本程序的动画同样是通过动画描述文件自定义的
      2. 动画文件的格式
         1. #... 该行为注释
         2. movie length fps定义视频基本信息，length为视频长度，以帧为单位，fps为每秒帧数
         3. id[-id] rotate center axis degree fS fE定义一个（一系列）物件的旋转运动，其中id为物件编号（物件在AllObjects中的编号），该编号可根据场景文件中的物件顺序确定（以0开始）。扩展为中括号内部分，即可知道从编号a到编号b直接的所有物件做其后的选择运动。Center为旋转中心，axis为旋转轴的方向向量，degree为选择度数（弧度），fS为起始帧数，fE为终止帧数。
         4. id[-id] rotateSelf degree fS fE定义一个自旋转运动，其中id[-id]、fS、fE同4.3.2.3中所述，degree为旋转角。该运动使物体绕自己中心做顺时针旋转运动（degree<0时逆时针旋转）。定义该运动主要用于解决运动中物体中心不确定的问题。
         5. id[-id] move pos fS fE定义一个平移运动，该运动使得物体在指定帧过后移动至pos。值得注意的是，该运动在每一帧都将计算一次移动距离，因此，不论过程中物体进行了何种其它运动，在结束帧时都将移动至pos处。请勿在同一结束帧定义多个移动至不同地点的move运动。
         6. id[-id] moveRadius center distance fS fE定义一个径向运动，运动方向为指定center与物体中心连线的方向，规定指向物体中心方向为正。
         7. id[-id] moveDirect direction distance fS fE定义一个固定方向的平移运动，direction为移动的方向，distance为移动的总距离
         8. back id fS fE定义了一个背景变换，在结束帧时将背景变换为第id各纹理图。该变换将自动实现渐变。
         9. camera u/f/v rotate/move/moveRadius/moveDirect …定义了相机的运动，u、f、v分别表示顶部、前方和视点的运动，其后的运动定义和物件的运动类似。注意，相机没有rotateSelf的定义。
   4. 加速

除使用八叉树加速外，对程序的初始化和运行均进行了优化与加速。其中，初始化部分实现了边分配物件边创建场景，使得物件分配占用更少的内存即可实现。在分配与追踪过程中使用多线程技术，使得花费时间更短。

* 1. 高维纹理

场景的球型纹理背景增加了视觉上的宏大感，对于大型场景的渲染非常有效。

* 1. 背景渐变

应用背景渐变，使得动画具有更多的观赏性与可创造性，例如demo中的天气变化。

* 1. 键盘控制

这个功能比较纠结……由于程序的性能问题，使得每一次按键后均需等待一段时间，短则几秒，长则几十秒，因此实用性能有所下降。虽然与最初的设想差距比较大，不过还是可以使用的。

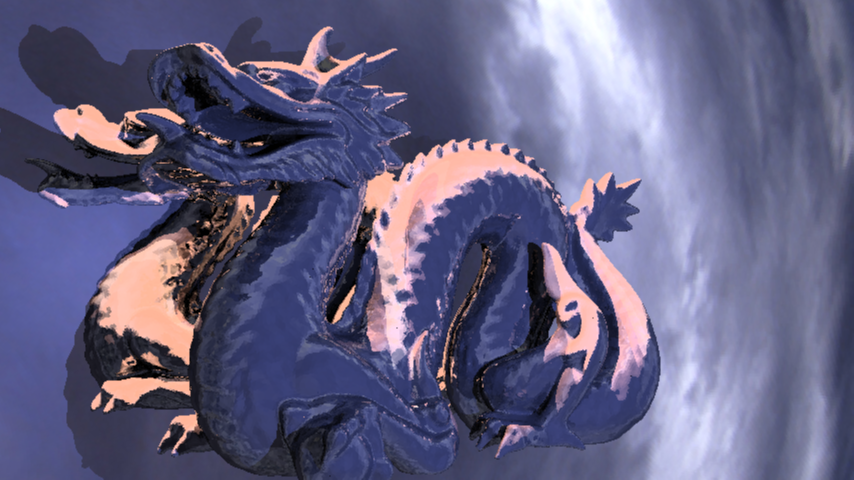
1. 程序运行结果
   1. 普通光线追踪与两种抗锯齿效果
      1. 普通光线追踪



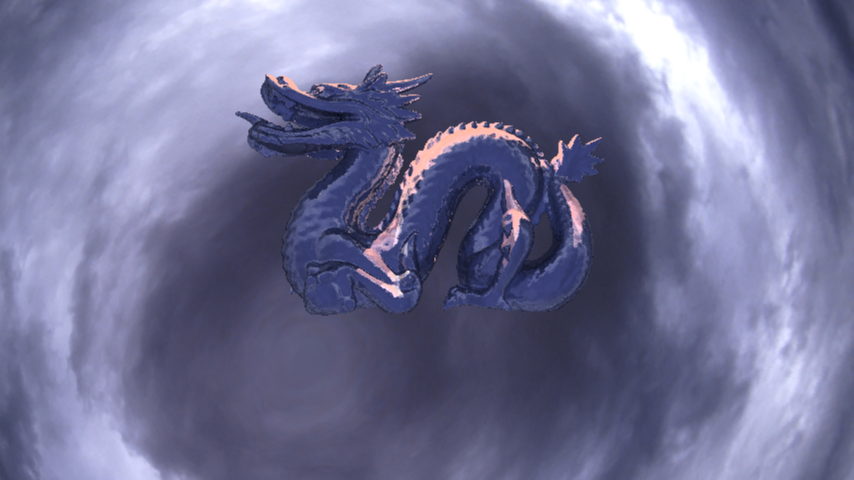
* + 1. FSAA4x抗锯齿



* + 1. Smooth平滑



* 1. 普通光线追踪——Dragon



* 1. 多obj
     1. 普通光线追踪



* + 1. FASS4x抗锯齿



* + 1. FASS4x抗锯齿+Smooth平滑



* 1. 动画

见打包附件

1. 总结与收获
   1. OpenCV
      1. openCV的版本不尽相同，在生成exe时应注意将对应版本的dll拷贝至exe文件所在目录。
      2. 在读取图像时，默认读取的图像为8位无符号整型，在运算时应转化为浮点型。
      3. 在储存图像时，应将32位无符号浮点型图像转化为8位无符号整型图像再进行存储。
      4. 输出图像时，cvSize的两个参数分别为列和行，而不是行和列。
   2. 遇到的困难
      1. 计算精度问题：在进行光线求交运算时，所得数据精度需根据具体情况进行调整，如球体求交精度在1e-3，而三角和平行四边形面片求交精度在1e-4。
      2. 空间占用问题：最初每个场景的物件列表是使用vector存储的，这导致在初始化时，若面片数目过大，内存会不够用。究其原因在于vector的连续空间存储性质，这使得内存中有许多空闲空间碎片。改为list结构后便更充分利用了空间。
      3. 全反射问题：最初在计算折射时并未考虑全反射，这使得对于立方体的情况，在入射角大于全反射角时，折射光线方向变得非常诡异，产生的景象也很奇怪。后来经过调试发现光线在立方体内发生了全反射，因此加入了全反射判断。
      4. 屏幕处于场景外的问题：有时候，在设置相机位置时设置有误，使得屏幕有一部分在场景外，此时在产生光线时需要对光线起点进行修正。在无法修正的情况下默认显示黑色。
      5. 多线程访问冲突问题：这在多线程加速部分已经提到，在维护物件的involvedScene列表是须加入互斥锁。
      6. 运动场景更新的优化问题：这在运动渲染加速部分也已经提到，通过只重新分配运动物体以达到运动场景更新的加速。
2. 参考资料
   1. 课件
   2. CSDN博客（具体参考的文章也记不住了，通常是直接用搜索引擎搜索的CSDN的博文）