计算机图形学大作业

CSG 实体建模和真实感绘制

张雯莉 5090379039 <OviliaZhang@gmail.com>

概述

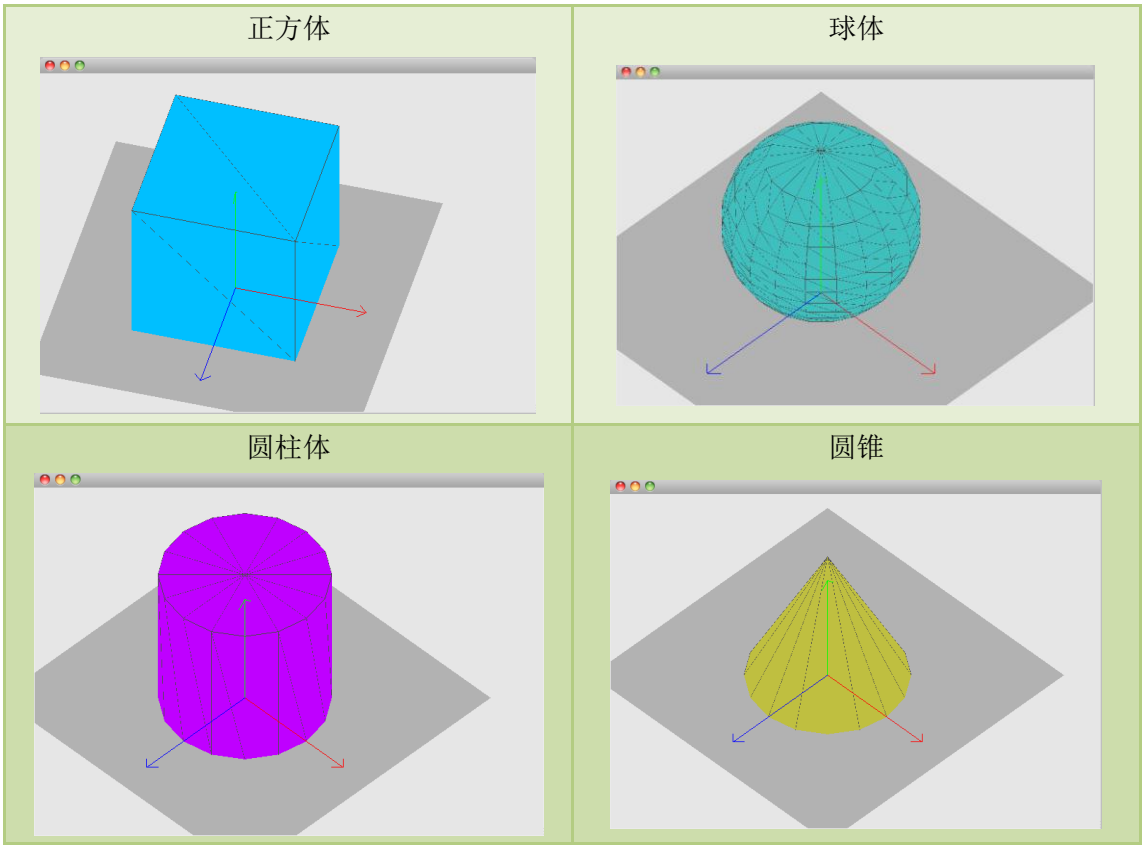
本作品使用 OpenGL 实现 CSG 实体建模和真实感绘制。完成内容包括：

1) 6 种 Primitive[A]建模；Primitive 和 Solid[B]的并操作；交互修改 Primitive 和 Solid 属性
2) 灯光设置；打开、关闭、隐藏光源；交互修改灯光的属性
3) 可交互编辑的纹理映射；不同 Primitive 对应不同的纹理映射
4) CSG 模型的存储和读取显示
5) 四视图；移动、放大缩小视图；ArcBall 旋转视图
6) 本作业报告、代码和可执行程序

具体说明

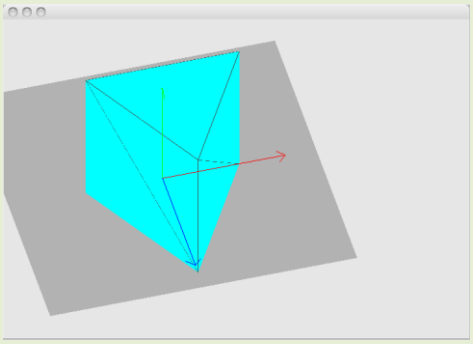
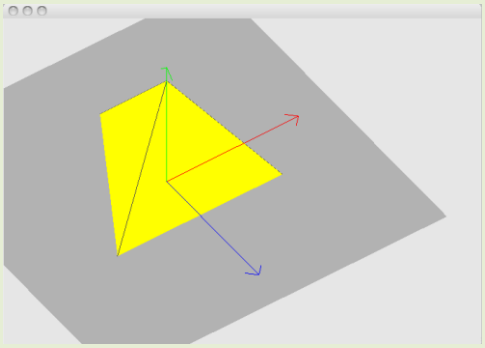
1. 体素建模

1.1. 6 种 Primitive 建模



[A] Primitive 指体素，即最基本的几何体，如：长方体、球体等。下同。

[B] Solid 由 1 个 Primitive 或者 2 个 Solid 组成。当其由一个 Primitive 组成时，该 Solid 是叶子节点；当其由 2 个 Solid 组成时，表示该 Solid 是其 2 个子 Solid 通过布尔操作得到的结果。

棱柱	金字塔体
	

完成 6 种 Primitive 的建模，分别是：长方体、球体、圆柱体、圆锥体、金字塔体、棱柱体。为了对点面有更好的控制，所有 Primitive 都采用三角面片存储点和面的信息。

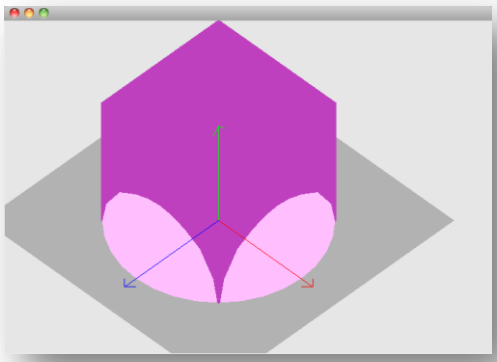
1.2. 并操作

并操作的对象是 2 个 Solid。由于 Solid 可以是只包含 1 个 Primitive 的叶子节点，也可以是包含 2 个 Primitive 和它们对应的布尔操作的非叶子节点，这样，并操作的对象将是一棵树。一个可能的结构是这样的：

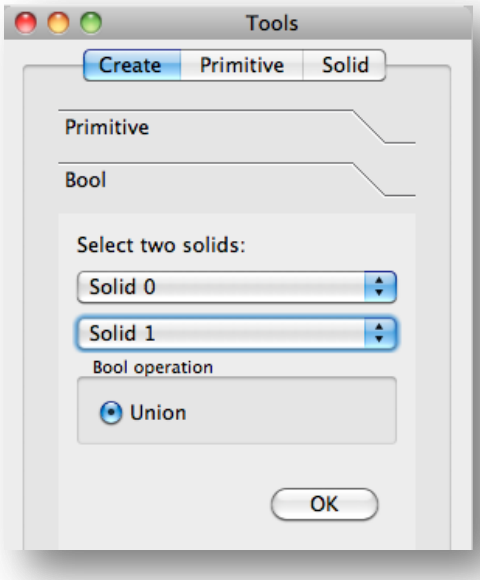
```
<Solid>
    <Solid>
        <Primitive />
    </Solid>
    <Solid>
        <Solid operation=”Union”>
            <Primitive />
            <Primitive />
        </Solid>
    </Solid>
</Solid>
```

建立并操作的方法：

a）建立两个基本体素（图示为 Cube 和 Sphere 的组合）



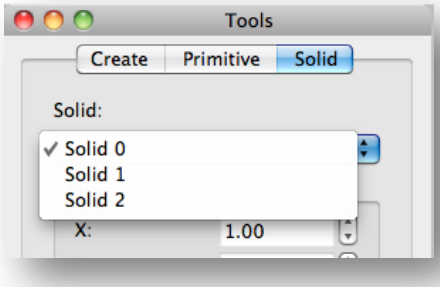
b) 在 Create 面板中选择 Bool



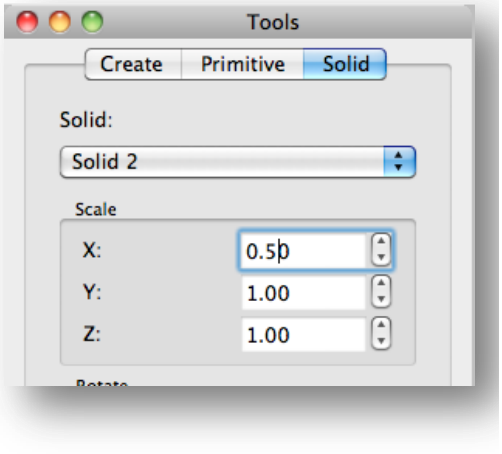
c) 选择基本体素和布尔操作（目前只有并操作）

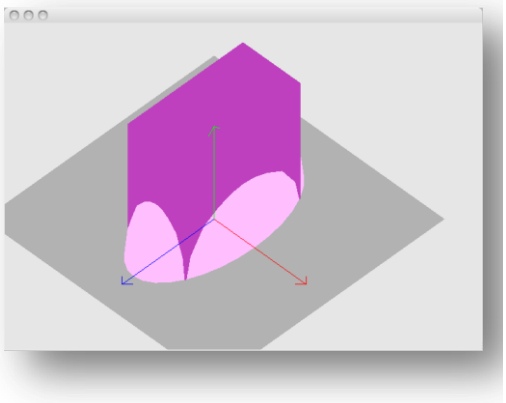
d) 点击 OK 按钮

e) 可以在 Solid 面板中发现多了一项，就是组合后的结果



f) 在 Solid 面板中的操作将作用于组合整体

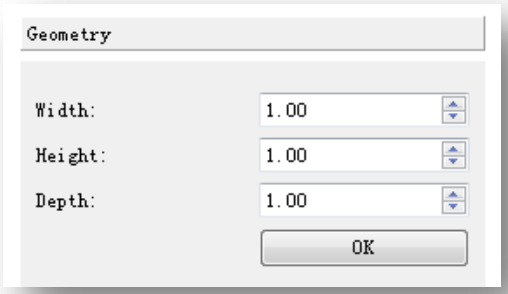




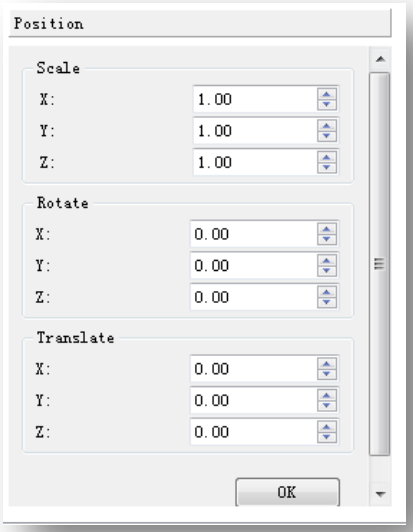
1.3. Primitive 和 Solid 属性

Primitive

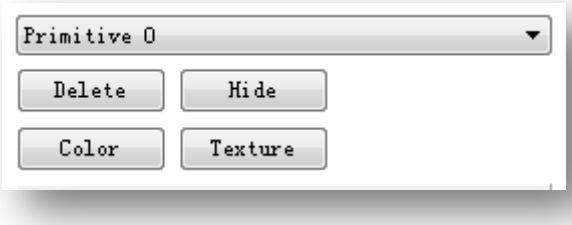
不同的 Primitive 对应不同的几何属性，如：长方体有长、宽、高；球体有半径、片数、层数等。可以在 Geometry 面板内修改几何属性。



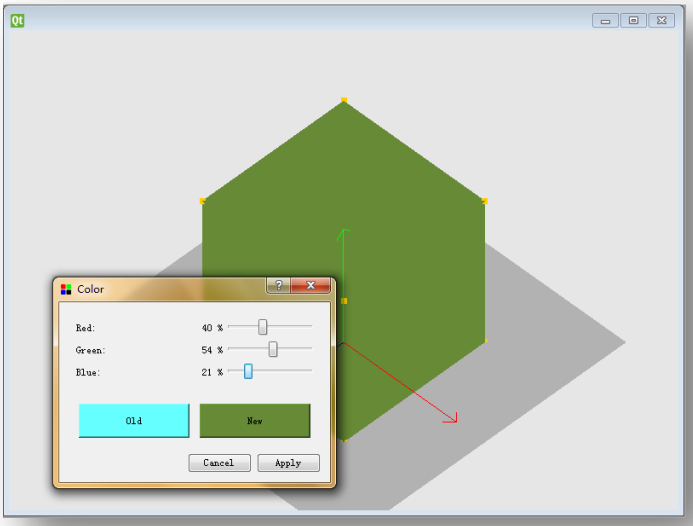
Primitive 有共有的属性：缩放、旋转、移动，可以在 Position 面板修改。



此外，还能修改 Primitive 的以下属性：

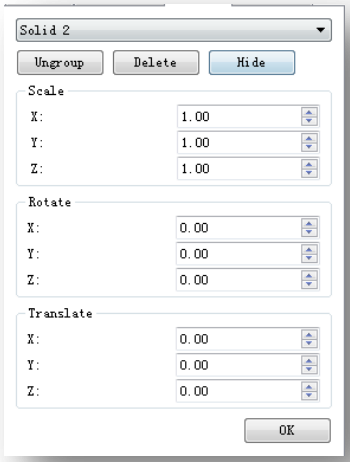


除了能删除 Primitive，还能将其暂时隐藏，这在编辑时也十分有用。Texture 属性将在纹理映射中详细说明。点击 Color 可以看到以下界面，拖动滚动条修改颜色。



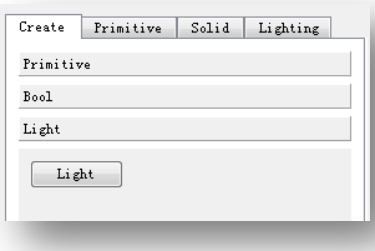
Solid

Solid 属性面板中，可以对 Solid 取消布尔操作、删除和对缩放、移动、旋转参数的修改，修改后会作用在 Solid 整体。

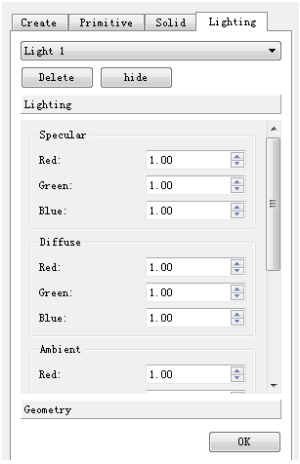


2. 灯光建模

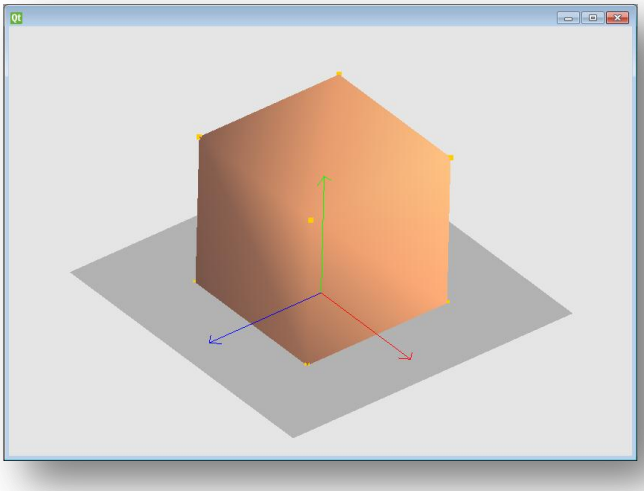
在 Create 面板中新建灯光：



在 Lighting 面板中修改灯光属性，包括：Specular、Diffuse、Ambient、SpotCutOff、SpotDirection。后两个参数可以用来控制聚光灯。

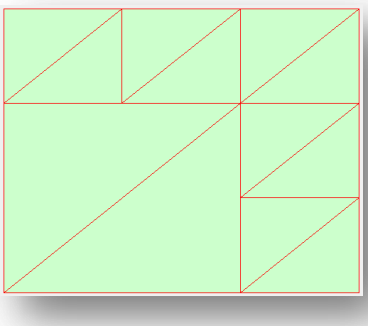


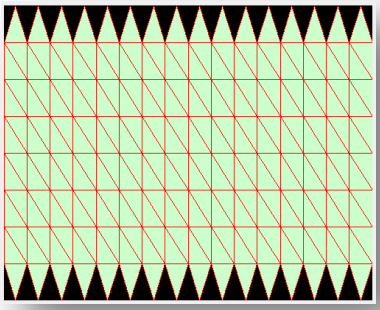
点击 Hide 能暂时关闭灯光，点击 Delete 将删除该灯光。



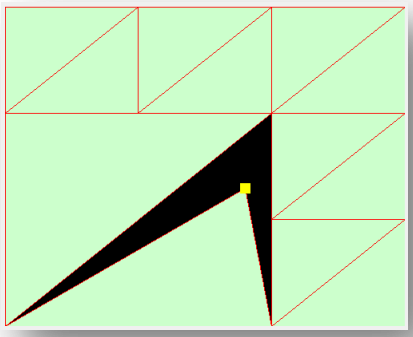
3. 纹理映射

点击 Primitive 面板中的 Texture 按钮可以修改纹理映射。在没有设置纹理的情况下，不同的 Primitive 都设置了默认的纹理映射方案。长方体和球体的默认纹理映射方案如下：



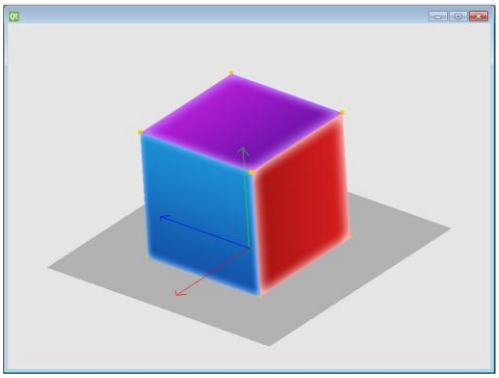


拖动点可以修改点的位置：

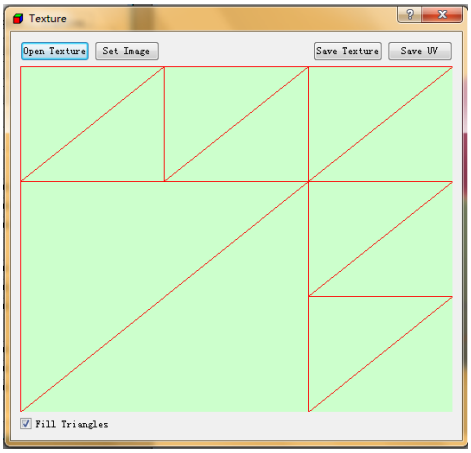


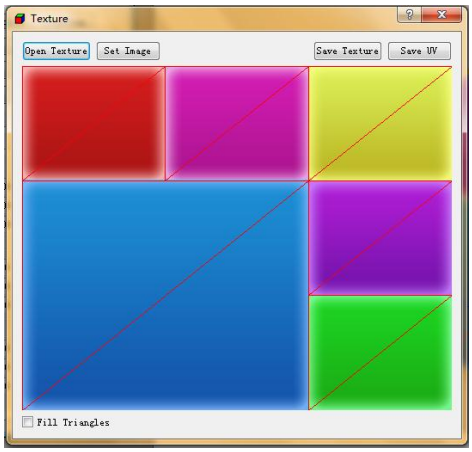
点击 Save Texture 可以将点的位置信息保存成 xml 文件，点击 Open Texture 可以打开之前保存的 XML 文件。注意，如果打开的 XML 文件中点的个数与当前 Primitive 的点的个数不匹配的话会报错，打开失败。

点击 Save UV 可以将纹理映射的图保存成图片，然后在 Photoshop 等软件中修改你要的纹理图片，并将其保存。下图中，左图为纹理图片，右图为贴图效果。



在纹理设置窗口，可以控制三角形可见不可见，这样能够更清楚地看清贴图在模型中的对应位置。下图中，左图为三角形可见，右图为不可见。





4. 模型存储

模型采用 XML 存储，将 Solid 的树形结构很好地与 XML 的树形结构结合。存储的信息除了 Solid 和 Primitive 的属性外，还包括了光照、纹理映射等信息。当用户修改了

模型并且新建或打开其他模型时，会提醒用户是否需要保存。

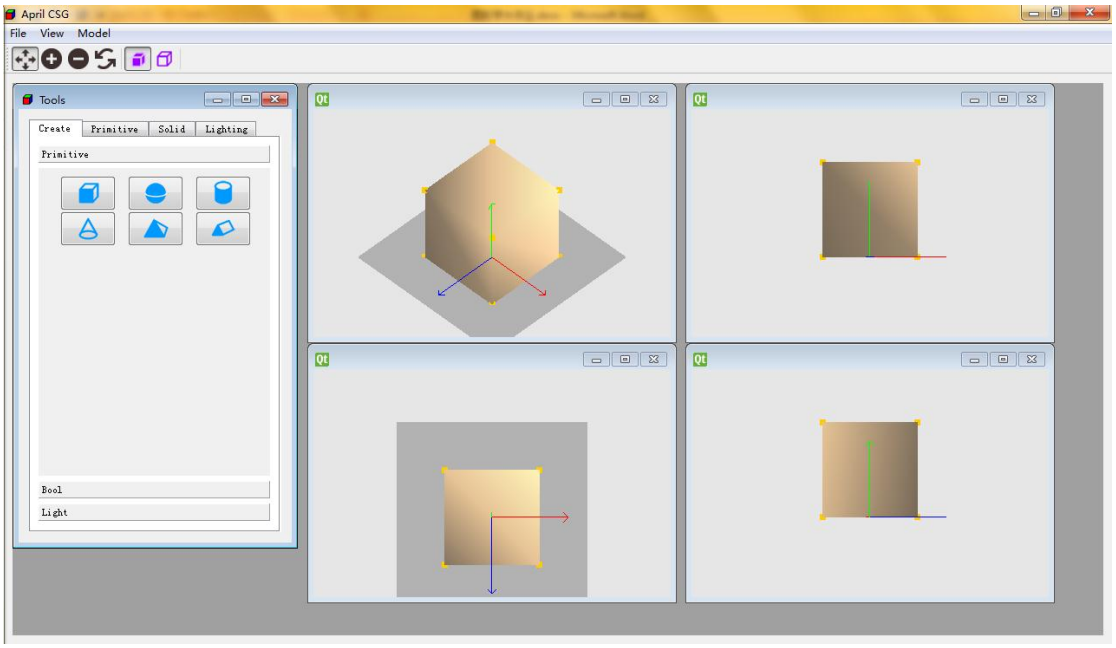
以下是一个典型的模型存储文件：

```
<!DOCTYPE April_Project_Model_Version1.3>
<AprModel>
 <solids>
  <solid operation="union" right="Solid 1" left="Solid 0" name="Solid 2">
   <solid name="Solid 0">
    <primitive width="1" height="1" type="cube" name="Primitive 0" depth="1">
     <color g="1" r="0.55" b="0.8"/>
     <texture uvFileName="C:/Users/Ovilia/Desktop/cube.atxt" fileName="C:/Users/Ovilia/Desktop/cube.png"/>
    </primitive>
   </solid>
   <solid name="Solid 1">
    <primitive slices="16" radius="0.7" height="1.3" type="cone" name="Primitive 1">
     <color g="0.99" r="1" b="0.42"/>
    </primitive>
   </solid>
  </solid>
  <solid name="Solid 3">
   <primitive slices="16" radius="0.5" type="sphere" stacks="8" name="Primitive 2">
    <scale x="2" y="1" z="1"/>
    <translate x="0" y="0.5" z="0"/>
    <color g="0.65" r="0.42" b="0.5"/>
   </primitive>
  </solid>
 </solids>
 <lighting>
  <light isOn="1" id="0">
   <position w="1" x="0" y="1" z="1"/>
   <ambient w="1" x="1" y="1" z="1"/>
   <diffuse w="1" x="1" y="0.8" z="0.8"/>
   <specular w="1" x="1" y="1" z="1"/>
   <direction w="1" x="-1" y="-1" z="-1"/>
   <cutOff value="180"/>
  </light>
 </lighting>
</AprModel>
```

5. 视图说明

5.1. 四视图

点击菜单栏 View -> 4 Views 打开四视图；点击菜单栏 View -> 1 View 关闭四视图。



5.2. 移动、放大缩小

菜单栏按钮分别对应移动、放大、缩小、旋转。点击后可以进行相应的操作。其中，移动功能可以做到随着鼠标的移动距离进行相应的视角移动；放大缩小是根据预设的值进行操作；旋转在下文进行详细说明。

5.3. ArcBall 旋转视图

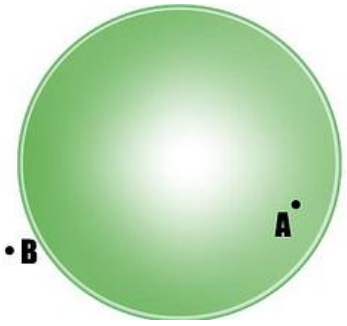
ArcBall 算法是一种经典的将二维坐标映射到三维坐标，从而实现三维场景的旋转的算法。通过本项目中的旋转功能，我学习了 ArcBall 算法的主要思想，并用其实现了三维场景的旋转。

算法的主要思想是把屏幕看成一个三维的球体从正面看上去的投影。把鼠标拖动的位置看成是对球体的旋转，将场景跟着这个球旋转得到结果。下面进行详细的说明，网上很少有详细说明 ArcBall 原理的，我综合理解后写了一篇博客对此进行详细的说明，以下内容摘自我的博客[C]。

分为以下四个步骤[D]：

1. 首先把按下鼠标和拖动鼠标的坐标记为 Q1，Q2，x 和 y 分别按屏幕大小缩放到[-1, 1]。如：Q1(100, 500), Q2(800, 600)，屏幕大小 1000x800。则缩放后得到的 P1(-0.8, -0.25), P2(0.6, -0.5)。之所以做这个映射完全是为了方便以后的计算，将在下文进行说明。

2. 把二维坐标转成三维的，这步是最关键的。现在我们可以把屏幕看成一个 xyz 都是[-1, 1]的球体了，球心在(0, 0, 0)处。

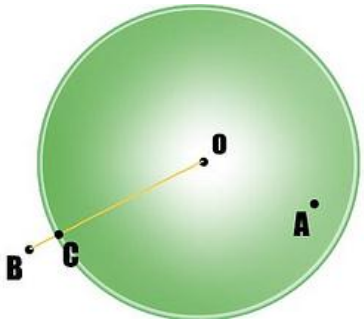


比如 A 和 B 是两个鼠标映射后的点，从前视图看，A 在球“上”（这里正确的理解是球壳上，而不是球体内部）；B 在球体外部。之所以说 A 在球壳上，是我们人为假设的，就是为了要对应到球体的转动。既然 A 在球壳上，我们就根据 x，y 值求的对应的 z 值（x、y、z 的平方和是 1，因为在球壳上）；对 B 而言，我们把它“就近迁移”到球壳上，那么球壳上离 B 最近的点是什么呢？从正视图看应该

[C] http://ovilia.blogbus.com/logs/214030655.html

[D] 参考：http://en.wikibooks.org/wiki/OpenGL_Programming/Modern_OpenGL_Tutorial_Arcball

是这样的：

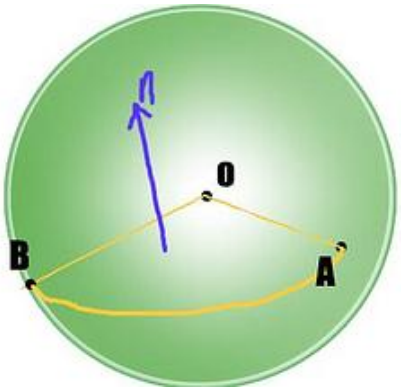


所以我们认为 C 点的 z 坐标是 0。所以三维坐标的计算方法：

```
double square = vec.x * vec.x + vec.y * vec.y;
if (square <= 1.0) {
    // if (x, y) is within the circle of radius 1
    // calculate z so that the modulus of vector is 1
    vec.z = qSqrt(1.0 - square);
} else {
    // if is out of the circle, do nomarlization
    // this vector is the nearest position on the circle
    // so that z is 0
    double length = qSqrt(square);
    vec.x /= length;
    vec.y /= length;
    vec.z = 0.0;
}
```

3. 接下来求旋转角。我们知道向量 A 点乘向量 B=|A||B|cos(alpha)其中 alpha 是向量夹角。根据前两步，我们能得到鼠标按下的位置 A 和拖动时当前位置在球上的坐标 B，现在我们想求出向量 OA 和 OB 的夹角。因为球的半径是 1，所以 |OA|=|OB|=1。那么 alpha=arccos(A 和 B 的点积)。这也就是为什么要把屏幕变换到-1 和 1 之间的原因了。

4. 我们知道 glRotatex 需要三个参数：一个旋转角和一个旋转轴对应的三个坐标。所以接下去我们就要求旋转轴。既然刚刚点积发挥过作用了，这次我们就要让叉乘出出风头了。向量 A 和 B 叉乘的结果是它们所在平面的法向量，也就意味着就是我们要求的旋转轴了。



但是由于我们只计算了鼠标按下的位置和当前鼠标位置的旋转效果，所以上一次旋转的效果在第二次按下鼠标时就消失了。记录下每次的旋转角和旋转轴显然不是一个好办法，因为旋转次数多了以后每帧都要调用非常多的 glRotatex 显然不合适。所以应该记录下每次旋转的旋转矩阵，然后利用矩阵乘法达到累积旋转的效果。

5.4. 总结

经过本学期计算机图形学知识的学习，使我知道了计算机提供绚丽画面的背后是那么多算法的支持。一开始，我对 OpenGL 十分底层，需要写很多代码才能出一点效果觉得很不好，但是，后来经过学习，我才明白底层也意味着更多的掌控，使得程序员能够对图形有更多的操作。完成这次作业我花了很多时间和精力，在这个过程中也体会到了图形学的美并对之产生兴趣。也非常感谢肖老师对我的帮助！