OpenGL 实验课程

计算机图形学　　指导老师：张宏鑫　　TA：利明

本次内容

- “太阳系”作业分析 & 上次实验作业分析
- OpenGL显示列表讲解 & 本次实验作业
- OpenGL Shading Language 简介
- 可选实验作业的布置

“太阳系”系统

- Build a Solar System
  - detailed computing steps
  - at least Earth, Moon and Sun
  - in OpenGL
- Bonus
  - implemented demo
  - orbit / trajectory simulation

“太阳系”系统

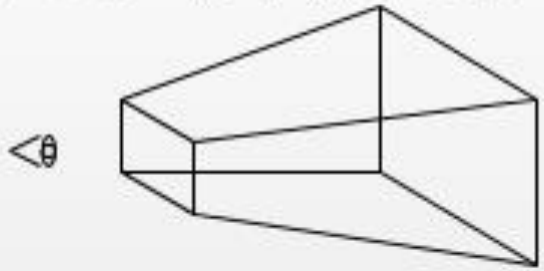
- 简化模型（自转，公转）
- 空间变换（旋转，平移）
- 星体、轨道的绘制方法
  - glPolygonMode(GL_FRONT, GL_LINE);
  - glBegin(GL_LINE_LOOP);
- 光照
- 贴图

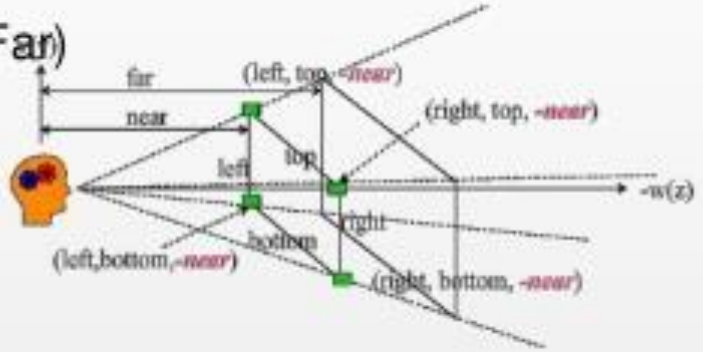
实验作业分析

- void glOrtho (double left, double right,
  double bottom, double top,
  double near_val, double far_val);
- void gluPerspective (double fovy, double aspect,
  double zNear, double zFar);

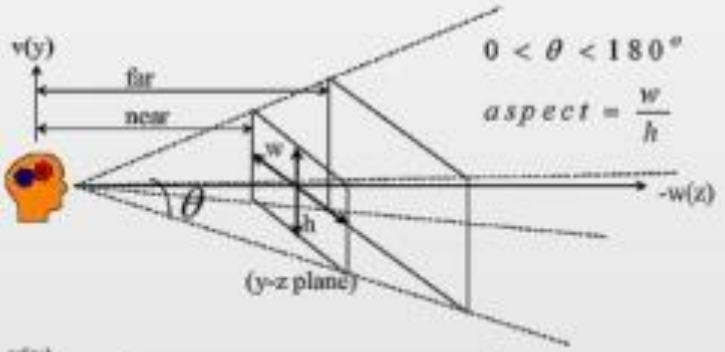
Part02 : Transformations-Viewing III

- glFrustum(left, right, bottom, top, zNear, zFar)

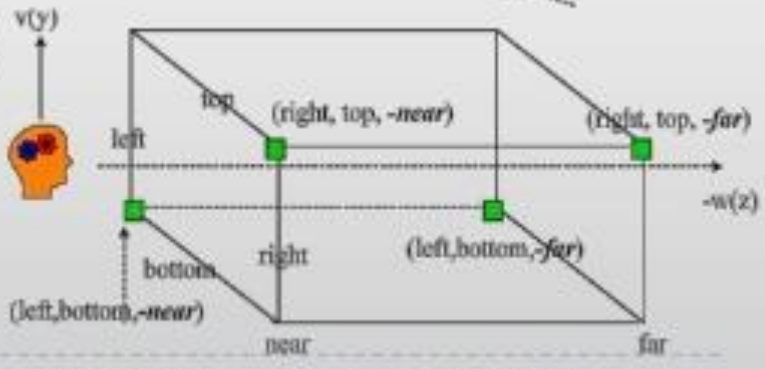




- gluPerspective(fovy, aspect, zNear, zFar)



- glOrtho(left, right, bottom, top, zNear, zFar)



OpenGL显示列表

- OpenGL显示列表（Display List）是由一组预先存储起来的留待以后调用的OpenGL函数语句组成的，当调用这张显示列表时就依次执行表中所列出的函数语句。
- 前面内容所举出的例子都是瞬时给出函数命令，则OpenGL瞬时执行相应的命令，这种绘图方式叫做立即或瞬时方式（immediate mode）。

OpenGL显示列表

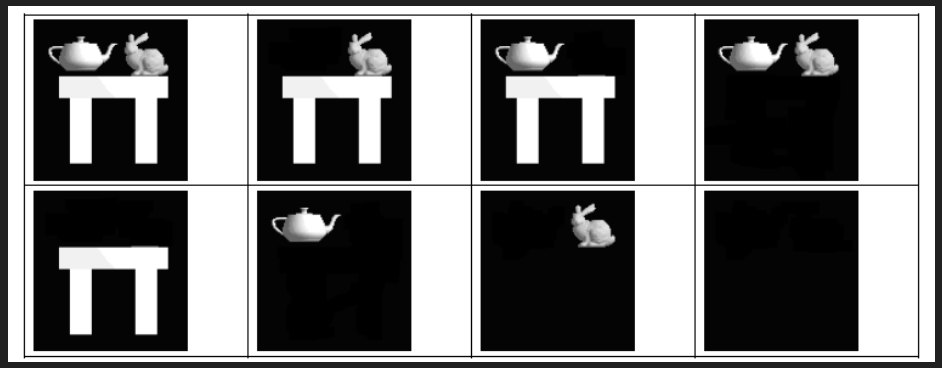
- OpenGL显示列表的设计能优化程序运行性能，尤其是网络性能。它被设计成命令高速缓存。一旦建立了显示列表，就不能修改它，采用显示列表方式绘图一般要比瞬时方式快。
- 当通过网络发出绘图命令时，显示列表驻留在服务器中，使网络的负担减轻到最小。在单机上，显示列表被处理成适合于图形硬件的格式，不同的OpenGL实现对命令的优化程度也不同。
- 旋转矩阵函数glRotate*()，若将它置于显示列表中，则可大大提高性能。因为旋转矩阵的计算并不简单，包含有平方、三角函数等复杂运算，而在显示列表中，它只被存储为最终的旋转矩阵，于是执行起来如同硬件执行函数glMultMatrix()一样快。
- 一般来说，显示列表能将许多相邻的矩阵变换结合成单个的矩阵乘法，从而加快速度。

OpenGL显示列表

- GLuint glGenLists (GLsizei range);
- void glNewList (GLuint list, GLenum mode)
  - GL_COMPILE
  - GL_COMPILE_AND_EXECUTE
- void glEndList (void);

本次实验作业

- 掌握OpenGL中显示列表的作用和使用方法，修改代码，通过键盘按键，控制各个物体（茶壶、桌子和兔子）的显示状态，生成以下图形：

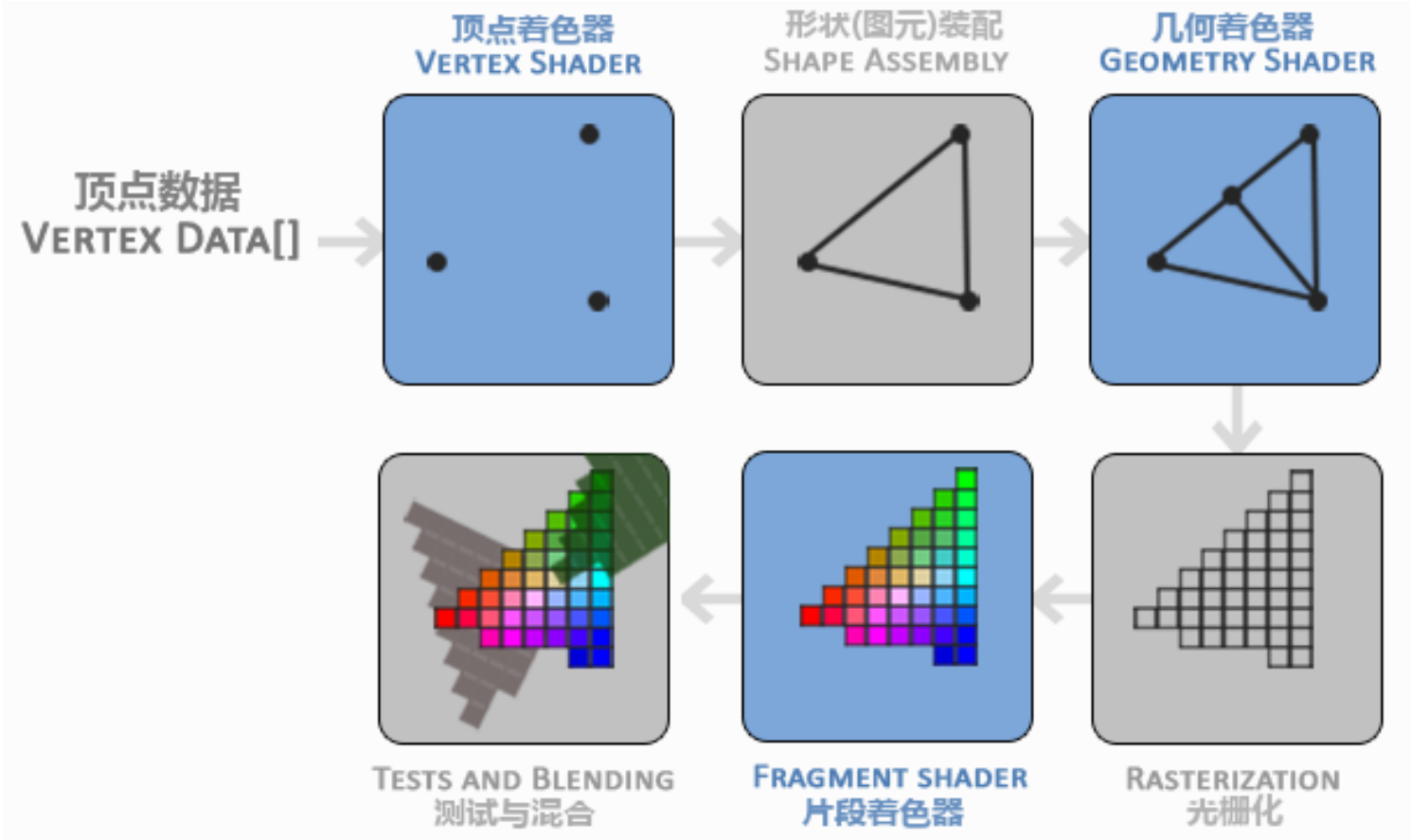


- 对各个物体分别用显示列表进行绘制，重复上面的交互控制。增加键盘按键来切换显示列表和非显示列表绘制方式，通过动画以及对FPS的理解和分析，理解显示列表对程序绘制性能的影响。

OpenGL Shading Language

- 图形渲染管线接受一组3D坐标，然后把它们转变为你屏幕上的有色2D像素输出。渲染管线可以被划分为几个阶段，每个阶段将会把前一个阶段的输出作为输入。
- 它们具有并行执行的特性，大多数显卡都有成千上万的处理核心，它们在GPU上为每一个（渲染管线）阶段运行各自的小程序，从而在图形渲染管线中快速处理你的数据。这些小程序叫做着色器(Shader)。

OpenGL Shading Language



简单的Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 position; // position变量的属性位置值为0

out vec4 vertexColor; // 为片段着色器指定一个颜色输出

void main()
{
    gl_Position = vec4(position, 1.0); // 注意我们如何把一个vec3作为vec4的构造器的参数
    vertexColor = vec4(0.5f, 0.0f, 0.0f, 1.0f); // 把输出变量设置为暗红色
}
```

简单的Fragment Shader

```
#version 330 core
in vec4 vertexColor; // 从顶点着色器传来的输入变量（名称相同、类型相同）

out vec4 color; // 片段着色器输出的变量名可以任意命名，类型必须是vec4

void main()
{
    color = vertexColor;
}
```

其他

- 下次可能会增加可选实验内容
- 推荐网站：https://learnopengl-cn.github.io