### 期中考试考试范围

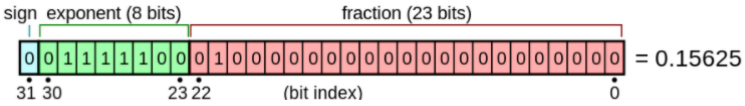
- C/C++编程基础：基本语法，计算精度，性能优化
- 渲染管线：基本组成部分，每部分的输入输出
- OpenGL基本语法：基本绘制语句，矩阵操作，程序结构
- 变换：除四元数外所有内容，注意内容理解而不仅是计算

# C/C++编程基础

## 基本语法

## 计算精度

- 基本类型
  - 字符类型：char(1)
  - 整型：int(4), short(2), long(4), long long(8)
  - 浮点类型：float(4), double(8)
- 浮点数的二进制表示
  - 单精度(float)




$$\text{sign} = +1$$
$$\text{exponent} = (-127) + 124 = -3$$
$$\text{fraction} = 1 + 2^{-2} = 1.25$$
$$\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625$$

## 性能优化

- 改进算法、数据结构，降低时间复杂度
- 在时间复杂度无法提高的情况下
  - 优化程序写法，减少不必要的运算，采用消耗较小的运算
  - 优化读写模式，充分利用缓存
- 性能优化举例
  - 强度折减(Strength reduction)：将开销高的运算替换为开销较低的运算
    - comparison: 1 clock cycle
    - (u)int add, subtract, bitops, shift: 1 clock cycle
    - floating point add, sub: 3~6 clock cycles
    - indexed array access: cache effect
    - (u)int32 mul: 3~4 clock cycles
    - floating point mul: 4~8 clock cycles
    - floating point division: 14~45 clock cycles
    - (u)int division, remainder: 40~80 clock cycles
  - 记录可重复使用的信息
  - 循环展开
  - 优化内存访问

    - CPU每次访问内存时，将加载包含该内存地址的一个cache line（通常为64B的倍数），此后对同一cache line的访问将通过缓存进行
    - cache line意味着访问连续内存空间的重要性，C/C++中数组为row-major，每行在内存中连续
    - 在矩阵乘法中，常见写法容易不自觉地使用column-major的访问模式，导致运行时间较长
    - 非对齐存储可能导致对一个数据的读取需要载入两个cache line，可利用__align__(n)强制对齐

# OpenGL基本语法

## 基本绘制语句

- glViewport(int x, int y, int width, int height)
    - x与y分别为视角左下角在当前窗口中的位置
    - width与height指明视口的大小
    - 视口的纵横比应当与当前观看的空间一致
- glVertex*()
    - ***=nt / ntv**, **n**-number(2,3,4), **t**-type(i=integer, f=float, etc.), **v**-vector
    - glVertex2i(5, 4)，指明坐标为(5, 4)的顶点，默认位于平面z=0，2指明顶点位置为2维向量，i指明向量中每一个分量为整型
    - glVertex3f(.25f, .25f, .5f)，指明坐标为(.25f, .25f, .5f)的顶点
    - double vector[3]={1.0, .33, 3.14159}; glVertex3dv(vertex); 指明坐标为(1.0, .33, 3.14159)的顶点，v指明输入为单个数组而非多个数字
- glBegin(primitiveType); ... ; glEnd();
    - 指明primitive type图元类型
    - primitiveType=GL_POLYGON, GL_POINTS等
- glPolygonMode(GLenum face, GLenum mode);
    - 指明多边形如何显示
    - face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
    - mode: GL_FILL, GL_LINE, GL_POINT
- glFrontFace(GLenum mode);
    - 指明如何判断前向面
    - mode: GL_CCW, GL_CW
- glCullFace(GLenum mode);
    - 指明剔除前向或（及）后向面
    - mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
    - 需要打开或关闭GL_CULL_FACE：glEnable(GL_CULL_FACE), glDisable(GL_CULL_FACE)
- glClearColor(0.0f, 0.0f, 0.0f, 0.0f); glClear(GL_COLOR_BUFFER_BIT);
    - 指定及应用背景颜色
- glColor*()
    - 指定物体颜色
    - glColor3f(1.0f, 1.0f, 1.0f);
    - glColor3i(255, 255, 255);
    - glColor3fv(color_array);
    - glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
- glFlush(); glFinish();
    - OpenGL命令并非立即执行，所有命令被置于一个command buffer中
    - 当绘制命令结束时，需要将这些命令发送至显卡

    - glFlush()异步执行（函数调用后立即返回），glFinish()同步执行（绘制结束后返回），两个命令均强制所有命令开始执行

## 矩阵操作

- glMatrixMode(GL_MODELVIEW); glMatrxiMode(GL_PROJECTION);
    - 建模矩阵和投影矩阵被置于两个矩阵堆栈之中
    - ModelView matrix (GL_MODELVIEW)
    - Projection matrix (GL_projection)
    - 在使用矩阵进行变化前，应先切换至相应的堆栈
- glRotate{fd}(angle, x, y, z);
    - 将current matrix乘以旋转矩阵，从而将场景中所有物体从原点到(x, y, z)的射线为旋转轴，逆时针旋转angle度
- glTranslate{fd}(x, y, z);
    - 将current matrix乘以平移矩阵，从而将场景中所有物体平移(x, y, z)
- glScale{fd}(x, y, z);
    - 将current matrix乘以缩放矩阵，从而使场景中所有物体沿三个坐标轴方向分别缩放(x, y, z)
- gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
    - 缺省视角为，摄像头位于原点，视线朝向-z方向，以+y为上
- glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble near, GLdouble far);
    - left, right, bottom, top指明了近裁剪平面(near)上的边界
    - (left, bottom, -near)与(right, top, -near)分别为近裁剪平面上的左下角与右上角顶点
    - far指明了远裁剪平面，near和far应取正值
- gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
    - fovy: y方向上的视野(field of view)宽度，单位为角度
    - aspect: 纵横比(width/height)
    - near与far: 近裁剪平面与远裁剪平面
- glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
    - 参数与glPerspective()很相似
    - (left, bottom, -near)与(right, top, -near)分别为近裁剪平面上的左下角与右上角顶点
    - 可以是任意不相等的两个值
- gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
    - (left, bottom)与(right, top)分别为近裁区域中左下角与右上角
    - 自动定义远近裁剪平面分别为-1.0及1.0

## 程序结构

- 头文件：#include<GL/gl.h> #include<GL/glu.h> #include<GL/glut.h>
- 静态库：opengl32.lib glu32.lib glut32.lib
- 动态库：opengl32.dll glu32.dll glut32.dll
- 基本程序结构
    - 非面向对象
    - 使用状态进行控制
    - 无限循环
- 回调函数工作流程
    - 主线程运行循环等待事件发生

        - 鼠标、键盘操作等
    - GUI框架提供指明回调函数的机制
        - 常使用函数指针作为参数传递
        - 将函数指针与特定事件联系
        - 当事件发生时调用对应的回调函数
    - 回调函数常带有参数
        - 事件调度器通过参数传递额外信息
        - 如鼠标操作的x与y坐标，键盘的键值等

# 渲染管线

## 基本组成部分

- 顶点处理：对每个顶点进行独立变换
- 光栅化：将由顶点定义的几何图元转成fragments
    - 每个fragment可以看做是按图像网格排列的三维空间中的一个像素
- Fragment处理：对每个fragment进行计算（颜色等）
- 合并：将fragment合并为像素

## 每部分的输入输出

- 顶点处理：处理物体空间Object Space或局部空间Local Space中的四维顶点坐标数据(x, y, z, w)，经过Model Matrix、View Matrix、Projection Matrix等矩阵变换和裁剪、透视除法等操作后，最终得到屏幕空间Screen Space中的二维顶点坐标数据(x, y)
- 光栅化：将屏幕空间Screen Space中的二维顶点转换为像素，也即将几何图元转变为片元fragment，一个像素可对应多个片元fragment
- Fragment处理：处理光栅化得到的像素，考虑光照、阴影等对物体的影响，计算得到像素每个片元fragment的最终颜色
- 合并：将片元fragment写进像素最终生成图像，替换使用离屏幕较近的fragment替换较远fragment，或按最远fragment到最近的顺序（或相反）进行渲染

# 变换

## 齐次坐标(homogeneous coordinate)

- OpenGL使用4维齐次坐标：$\mathbf{P} = [x, y, z, w]^{\mathrm{T}}$
- 当w为0时，P表示一个向量
- 当w不为0时，P表示一个点，其三维坐标为

$$x \leftarrow x/w, \quad y \leftarrow y/w, \quad z \leftarrow z/w$$

- 所有标准转换（旋转，平移，缩放）都可以被表示为4*4的矩阵乘法

## 平移变换

- 将点从一个位置移动至另一个位置
- $P^{'} = P + d$
- 由向量d决定，具有3自由度(x, y, z)
- 在齐次坐标中
    - $P = [x, y, z, 1]^T \rightarrow P^{'} = [x^{'}, y^{'}, z^{'}, 1]^T$
    - $d = [d_x, d_y, d_z, 0]^T$

$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{TP} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} y + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} z + \begin{bmatrix} d_x \\ d_y \\ d_z \\ 1 \end{bmatrix} 1 = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix}$$

### 2D旋转

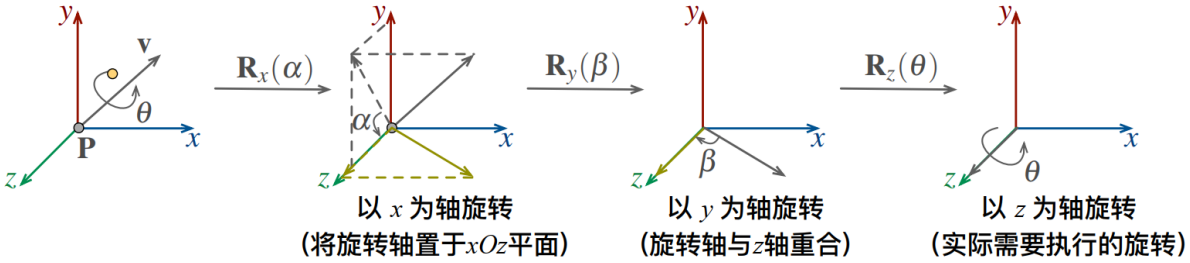
- 以原点为旋转轴，将向量逆时针旋转$\varphi$
  - 若该向量沿x轴：$\mathbf{v} = (r, 0)$，$\mathbf{v}' = (r \cdot \cos\varphi,\ r \cdot \sin\varphi)$
- 对任意长度为r的向量，将向量逆时针旋转$\theta$：$\mathbf{v} = (x, y) = (r \cdot \cos\varphi,\ r \cdot \sin\varphi)$
  - $v' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
- 以任意点为轴进行旋转：
  - $\mathbf{T}(-\mathbf{P})$
  - $\mathbf{R}(\theta)$
  - $\mathbf{T}(\mathbf{P})$

### 3D旋转

- 以x, y, z轴为旋转轴
  - 以z轴为旋转轴 $\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - 以x轴为旋转轴 $\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - 以y轴为旋转轴 $\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- 以过原点的任意向量为轴旋转

$$-\mathbf{R_V}(\theta) = \mathbf{R}_x(-\alpha)\mathbf{R}_y(-\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)$$




- 以不过远点的任意向量为轴旋转
  - 平移坐标系，使旋转轴经过原点
  - 旋转
  - 平移坐标系，回到原始位置
  - 变换矩阵 $\mathbf{M} = \mathbf{T}(\mathbf{P_f})\mathbf{R}(\theta)\mathbf{T}(-\mathbf{P}_f)$

## 缩放

- $P' = SP$
- $x' = S_x x,\ y' = S_y y,\ z' = S_z z$
- 变换矩阵 $\mathbf{S}(\mathrm{S}_x, \mathrm{S}_y, \mathrm{S}_z) = \begin{bmatrix} \mathrm{S}_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

## 错切

- 产生形状的变形
- 举例：沿x轴的错切 $x' = x + a \cdot y,\ y' = y,\ z' = z$