Lecture Notes 2024

Computer Graphics

Zhihao Li

Computer Science and Technology, Xidian University

zhihaoli@stu.xidian.edu.cn

Contents

Chapter 1

Writing Usage

1.1 Theorem Boxes

Definition 1.1.1: Definition Topic

Definition Statement

Theorem 1.1.1 Theorem Name

Theorem Statement

Corollary 1.1.2 Corollary Name

Corollary Statement

Lemma 1.1.3 Lemma Name

Lemma Statement

Claim 1.1.1 Claim Name

Claim Statement

Example 1.1.1 (Example Name)

Example explained

Chapter 2

Background

2.1 发展历史

- 上世纪 50 ~ 60 年代（1950-1969），阴极射线管（CRT）的出现促使了计算机图像学的兴起。阴极射线管，是一种用于显示系统的物理仪器，它是利用阴极电子枪发射电子，在阳极高压的作用下，射向荧光屏，使荧光粉发光，同时电子束在偏转磁场的作用下，作上下左右的移动来达到扫描的目的。

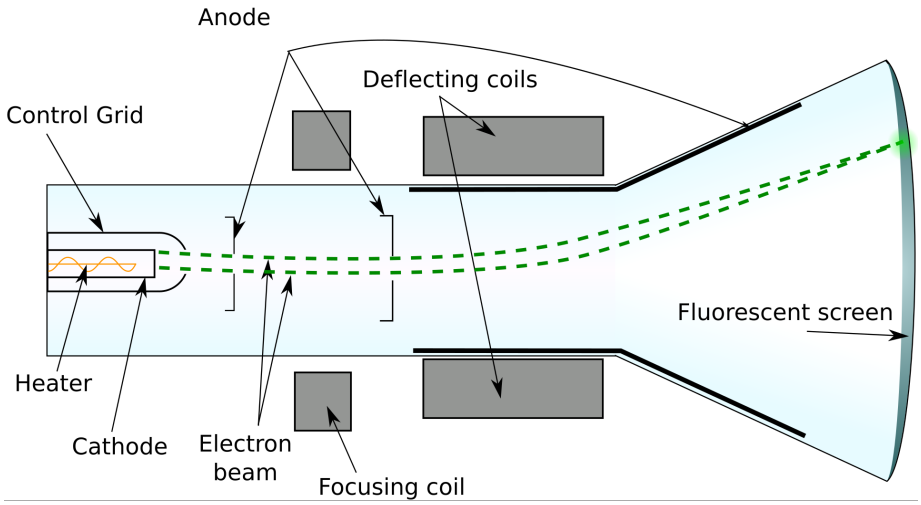


Figure 2.1: 阴极射线管中，电子束在磁场的作用下偏转

- 1950 年，第一个绘制出来的图形是由美国测绘员 Ben Laposky 通过操控模拟电子束在显示屏上呈现，这是最早的图形绘制设备，第一个通过阴极管绘制在计算机中的图形。
- 1959 年，美国通用和 IBM 公司发明了第一台工业 CAD 系统，帮助工程师进行车辆的设计，从此 CAD 逐渐发展起来。
- 1960 年，计算机图形学（Computer Graphics）是波音公司的工程师 Fetter 最早提出的。该概念主要用来描述飞机驾驶室模拟设计。

- 1963 年，MIT 博士生的 Ivan Sutherland （后获图灵奖，Coons 奖（图形学领域终身成就奖））设计了 Sketchpad，被公认为计算机图形学的起源。Sketchpad，赋予计算机图形处理的能力，使计算机拥有人类右脑的功能。

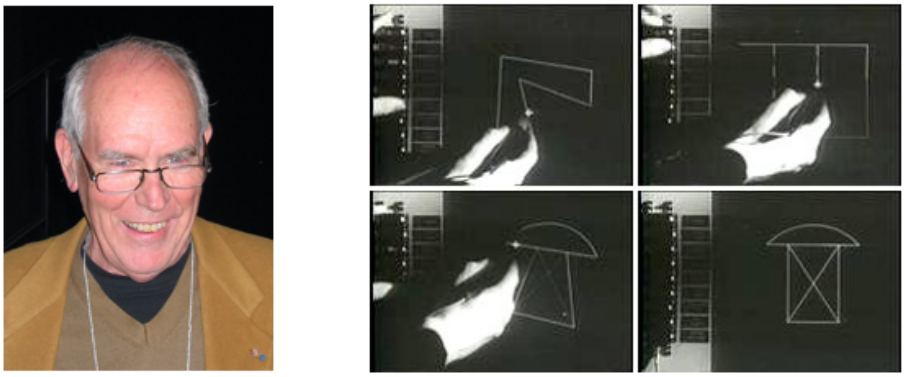


Figure 2.2: Ivan Sutherland 以及设计的 Sketchpad

Figure 2.3: 人类左脑符号处理以及右脑图像处理的能力

2.2 研究问题

1. 自然的三维交互技术：实现更灵活的三维交互输入，从传统鼠标、单点触摸、多点触摸的二维接触式交互方式转变为基于手势、姿态等的三维非接触式交互方式。
2. 直观的三维编辑技术：实现艺术家模式的三维模型制作与编辑，通过画笔手绘等常见且直观的艺术创作手法，对三维模型进行建模或修改。
3. 富有感情的机器人技术：实现具有人类相似情感表现的机器人，用人工的方法和技术赋予计算机或机器人以人类式的情感，使之在工作时具有表达、识别和理解喜乐哀怒，模仿、延伸和扩展人的情感的能力。
4. 绘画辅助技术：学习和控制绘制过程以及理解绘制物的语义，通过直线、曲线、笔画、素描等不同形式的绘画工具，让计算机更加智能地辅助进行人工参与的绘画创作。

5. 数学辅助技术：实现更便捷的数学符号和公式的输入、输出，通过图形方式辅助论文编辑、数学软件、数学手写板、数学推理等。
6. 海量数据的可视化技术：海量数据的视觉感知，借助面向海量数据的可视化技术，从根本上改变人们表示、分析和理解海量复杂数据的方式。
7. 虚拟导游技术：将虚拟角色融入吃、穿、住、用、行的虚拟现实场景，利用计算机提供的智能导游服务取代人力服务，将能大大提高服务质量。
8. 高沉浸感技术：真实感的虚拟环境沉浸式体验和交互，以虚拟洞穴、虚拟办公室、VR 眼镜等为代表的虚拟现实技术，需要使用户专注在当前的目标情境下感到愉悦和满足，而忘记真实世界的情境。
9. 增强现实技术：叠加虚拟物体到真实世界，在屏幕上把虚拟世界套在现实世界并进行互动，使用户就不必在其他设备上查看相关信息，以便腾出双手进行其他任务。
10. 混合现实技术：形成与真实世界无缝融合的高品质的完全沉浸式虚实融合的环境，通过交互构建现实世界、虚拟世界和用户之间的信息回路。

2.3 基本概念

计算机图像学研究真实或者虚拟世界的图形表示以及人机交互，是涉及计算机、数学、物理等学科的交叉领域。

Claim 2.3.1 Core Content

- 建模：在计算机中通过几何、图像、视频等形式构建和表示物体模型；
- 绘制：通过可视媒介（图像、视频、全息等）呈现计算机中存储的物理模型；
- 交互：通过计算机输入、输出设备，以有效的方式对计算机中的模型进行操作；

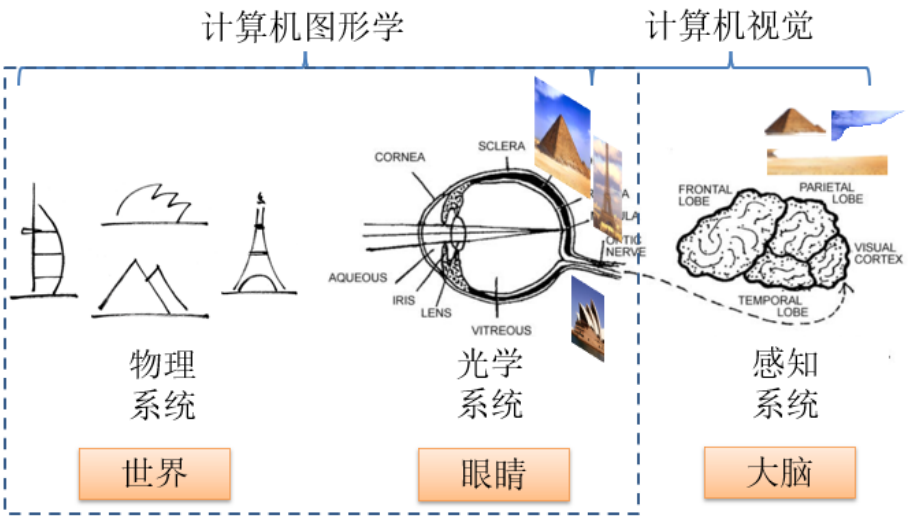


Figure 2.4: 计算机图像学与计算机视觉

Chapter 3

Basic Knowledge

3.1 图形与图像

Definition 3.1.1: 图形

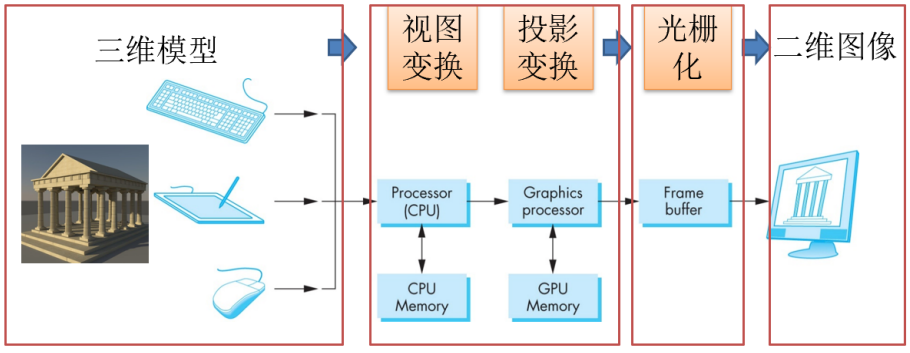
由点、线、面等基本几何元素作为“图元”构成，通过建模、测量等方式获取。

Definition 3.1.2: 图像

由像素构成，通过照相、扫描等方式获取。

Definition 3.1.3: 光栅化

二维和三维图形需要通过光栅化转化为图像进行屏幕显示，光栅化是图形学中一个重要的概念，它是将几何图形转化为像素点阵的过程。



计算机图形学系统

Figure 3.1: 图形流水线

3.2 计算机图形学系统

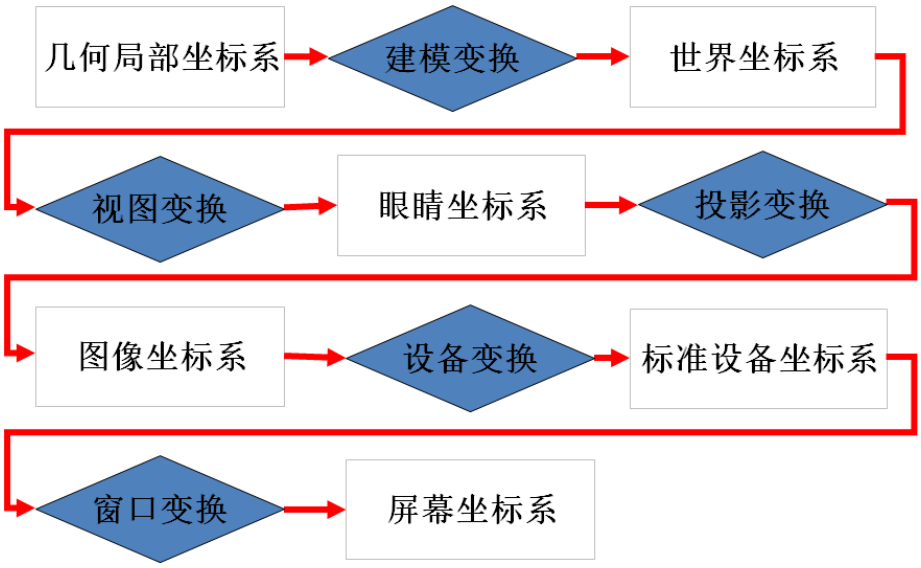


Figure 3.2: 计算机图形学系统

Chapter 4

Review of Linear Algebra

4.1 Vector Multiplication

4.1.1 Dot(Scalar) Product

点积又称为标量积 (Scalar Product) / 数量积，是两个向量对应元素乘积之和，通常被称为欧几里得空间的内积（或投影积）。

- 代数定义

  对于两个向量 $\vec{a} = [a_1, a_2, \cdots, a_n], \vec{b} = [b_1, b_2, \cdots, b_n]$ 的点积定义为，

  $$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{4.1}$$

  如果是 $\vec{a}, \vec{b}$ 都是列向量，可以进一步写为向量乘积的形式:

  $$\vec{a} \cdot \vec{b} = \vec{a}^T \vec{b} \tag{4.2}$$

- 几何定义

  在欧几里得空间中，向量被定义为具有大小和方向的几何对象，其点积定义为:

  $$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\theta \tag{4.3}$$

- 运算性质

  $$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a} \tag{4.4}$$

  $$\vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c} \tag{4.5}$$

  $$(k\vec{a}) \cdot \vec{b} = \vec{a} \cdot (k\vec{b}) = k(\vec{a} \cdot \vec{b}) \tag{4.6}$$

- 常见用法

  计算向量长度：向量的长度等于向量本身的点积的平方根，即 $\|\vec{a}\| = \sqrt{\vec{a} \cdot \vec{a}}$

  计算向量夹角：两个向量夹角的余弦等于它们的点积与它们长度的乘积的商，即 $\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$

判定向量正交：如果两个向量正交，则点积 $\vec{a}\cdot\vec{b}=0$

计算向量投影: $\vec{b}$ 在 $\vec{a}$ 上的投影，$\vec{b}_{\perp}=k\vec{a}=\|\vec{b}\|\cos\theta\vec{a}$

测量向量 close 程度：$\vec{a}\cdot\vec{b}>0$ 表示两个向量同向，$\vec{a}\cdot\vec{b}<0$ 表示两个向量反向，数值表示同向或反向的程度.

4.1.2 Cross(Vector) Product

叉积又称为向量积，其结果是一个向量，与两个初始的向量相互正交，方向可以由右手定则/右手螺旋法则确定。

- 基本定义

  叉积 $\vec{a}\times\vec{b}$ 定义为与 $\vec{a}$ 和 $\vec{b}$ 都垂直（正交）的向量 $\vec{c}$，即

$$\vec{a}\times\vec{b}=\|\vec{a}\|\|\vec{b}\|\sin\theta\vec{n} \tag{4.7}$$

  其中，$\vec{n}$ 是垂直于包含 $\vec{a}$ 和 $\vec{b}$ 的平面的单位向量.

- 运算性质

$$\vec{x}\times\vec{y}=-\vec{y}\times\vec{x} \tag{4.8}$$

$$\vec{a}\times\vec{a}=0 \tag{4.9}$$

$$\vec{a}\times(\vec{b}+\vec{c})=\vec{a}\times\vec{b}+\vec{a}\times\vec{c} \tag{4.10}$$

$$\vec{a}\times(k\vec{b})=k(\vec{a}\times\vec{b}) \tag{4.11}$$

- 图形学中用法

  确定左与右，以及确定内和外。如下图所示，根据右手螺旋定则可以判定，由 $\vec{a}\times\vec{b}$ 方向与 z 轴方向一致，因此可以判断 $\vec{b}$ 在 $\vec{a}$ 的左边，同理由 $\vec{b}\times\vec{a}$ 方向与 z 轴方向相反，因此可以判断 $\vec{a}$ 在 $\vec{b}$ 的右边。其中主要的判断依据是，右手螺旋定则是从右手边向左手边旋转，以此确定 z 轴的方向，如果有一个向量与 z 轴方向一致，那么它一定遵循右手螺旋定则。

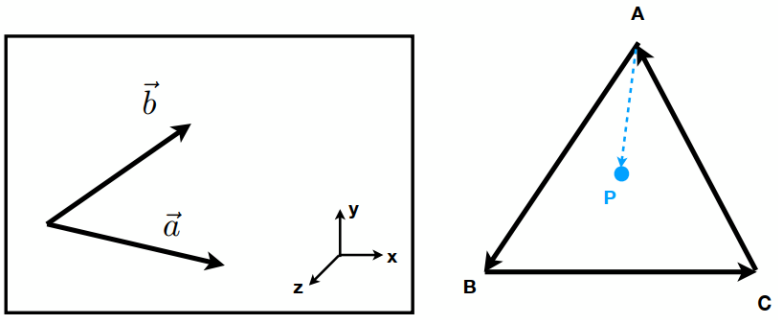


Figure 4.1: 叉积在图形学中的应用

对于三角形中，按照向量 $\vec{AB}\times\vec{AP}$ 确定 P 点在 $\vec{AB}$ 的左侧还是右侧，同理依次确定 P 点相对于 $\vec{BC}$ 以及 $\vec{AC}$ 的位置，如果 P 点都在三条边的左侧或右侧，则可以判断 P 点是在三角形的内部，否则在三角形的外部。

4.1.3 Orthonormal Coordinate Frames

正交坐标系，任何一组包含 3 个三维向量的向量组，如果满足如下条件，即可作为正交坐标系的标准正交基，

$$\|\vec{u}\| = \|\vec{v}\| = \|\vec{w}\| = 1 \tag{4.12}$$

$$\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{w} = \vec{u} \cdot \vec{w} = 0 \tag{4.13}$$

$$\vec{w} = \vec{u} \times \vec{v}(right - handed) \tag{4.14}$$

那么，对于向量空间（R^3）中的任意向量 $\vec{p}$ 都可以进行正交分解为，

$$\vec{p} = (\vec{p} \cdot \vec{u})\vec{u} + (\vec{p} \cdot \vec{v})\vec{v} + (\vec{p} \cdot \vec{w})\vec{w} \tag{4.15}$$

Chapter 5

Transformation

对于变换在图形学中主要应用在建模中相机的变换、模型的旋转、模型缩放以及 3D 到 2D 的投影变换。

5.1 2D Transformations

对于二维场景中的物体变换，在初高中教科书中都有涉及这种二维直角坐标系下的物体变换，包括平移、旋转、对称、缩放等操作，尤其是高中学过的仿射变换，在图形学中也会有用到。相比于高中所学，图形学中的 2D 变换更加侧重于利用矩阵表示图形变换。

5.1.1 Scale

缩放是指对于一个图形的所有维度进行伸缩，主要表现为延展和紧缩，具体需要关注于缩放矩阵，

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.1}$$

其中，s_x 与 s_y 分别表示对 x 和 y 维度施加相同或不同的缩放系数。

5.1.2 Reflection

对称是指将图形沿着某一维度进行镜像对称，例如如果按 y 轴做镜像对阵，其对应的对称矩阵为，

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.2}$$

5.1.3 Shear

错切是一种比较新颖的图形变换方式，更加直观地讲可以认为这种变换方式是一种图形的倾斜，例如对于下图中的图形，主要是沿 x 轴进行倾斜，y 坐标保持不变，其对应的错切矩阵为，

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.3}$$

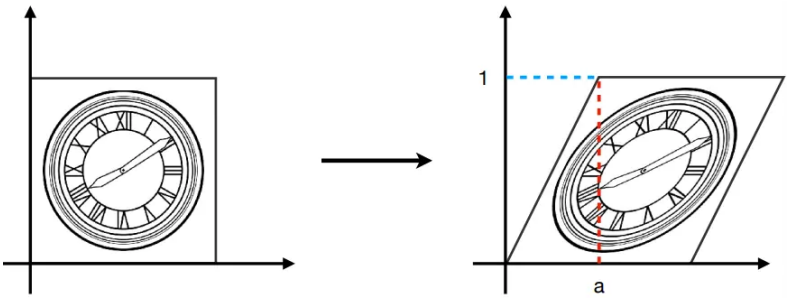


Figure 5.1: 错切图形变换

5.1.4 Rotate

旋转就是将图形绕某个中心以顺时针或逆时针方向进行旋转，但是在图形学中，默认的旋转方式都是图形绕坐标系原点以逆时针方向进行旋转，这样的旋转方式是最简单直观的，而其他更加复杂的旋转方式都是通过图形变换分解操作来实现，因此掌握默认的旋转方式即可。

默认的图形旋转方式如下图所示，其对应的旋转矩阵为，

$$\boldsymbol{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.4}$$

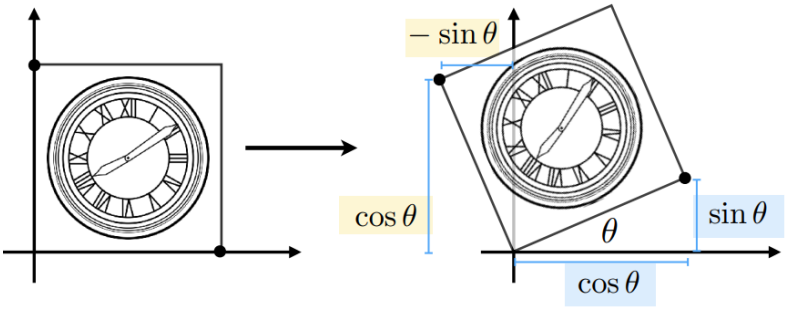


Figure 5.2: 默认的图形旋转方式

5.1.5 Translation

平移操作是指将图形按某个方向平行移动一段距离，实际上是对图形的坐标都增添一个位移量，具体表述为

$$x' = x + t_x \tag{5.5}$$

$$y' = y + t_y \tag{5.6}$$

5.1.6 Homogenous Coordinates

我们希望利用矩阵的性质找到这些图形变换的统一表示形式，即 $\boldsymbol{x}' = \boldsymbol{M}\boldsymbol{x}$. 但是，平移操作是对原始坐标进行增量操作，无法表示成矩阵相乘的形式，即

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.7}$$

解决方法就是齐次坐标（Homogenous Coordinates），在计算机图形学和几何学中，齐次坐标常用于表示平移、旋转和缩放等几何变换，使得这些变换可以通过矩阵运算来统一处理。齐次坐标引入了一个额外的维度，以便在同一框架内表示线性和仿射变换。

齐次坐标增添了额外的 w 维度，分别表示坐标点和向量的形式，

- 2D Point = $(x, y, 1)^T$
- 2D Vector = $(x, y, 0)^T$

当用齐次坐标重新表示平移操作时，可以得到统一的表示形式

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} \tag{5.8}$$

齐次坐标的有效性解释: 对于 Vector 而言，其 w 维度为 0 可以保持向量线性运算的性质，而对于 Point 而言，其 w 维度为 1 可以构造出向量以及获得坐标点，具体表示如下：

- vector + vector = vector
- point - point = vector
- point + vector = point
- point + point = middle-point

在齐次坐标中，对于每个坐标点更加广义的定义为，

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}, w \neq 0 \tag{5.9}$$

5.1.7 Affine Transformations

仿射变换主要包括线性变换以及平移，即 affine map = linear map + translation，公式表示为：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.10}$$

如果采用齐次坐标表示，仿射变换可以统一为：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.11}$$

具体而言重新表示上述各种图形表换方式对应的变换矩阵：

$$\boldsymbol{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

5.2 Composing and Decomposing Tranforms

合成和分解图形变换方式能够让我们形成更加复杂的图形变换方式，以及对于复杂变换分解为基本变换的叠加，

5.2.1 Composing Transforms

假设有一组仿射变换序列，$A_1, A_2, A_3, \cdots$，可以通过矩阵相乘的形式将这些仿射变换合成更加一个复杂的变换，然后再对坐标点进行变换，这有助于提高处理性能，

$$A_n(\cdots A_2(A_1(x))) = \boldsymbol{A_n} \cdots \boldsymbol{A_2} \cdot \boldsymbol{A_1} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.13}$$

其中由于矩阵乘法的性质，图形变换的顺序不可调换，可以通过结合预先处理出变换序列的变换矩阵，这样简洁的变换形式的好处主要源自采用齐次坐标统一了各种图形表换方式。

5.2.2 Decomposing Transforms

特殊的图形旋转方式: 当图形不再围绕坐标系中心按照逆时针旋转时，我们可以将这种复杂的旋转方式分解为基本的旋转方式，对应的矩阵表示形式为: $\boldsymbol{T(c)} \cdot \boldsymbol{R}(\alpha) \cdot \boldsymbol{T}(-c)$

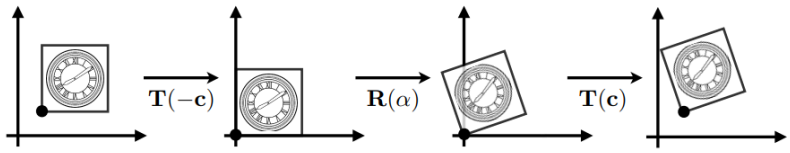


Figure 5.3: 复杂图形旋转方式的分解

5.3 3D Transformations

5.3.1 Homegeneous Coordinates

在建立了 2D Transforms 的齐次坐标变换后，可以直接得到对应于 3D Transforms 的齐次坐标表示，

- 3D point = $(x, y, z, 1)^T$
- 3D vector = $(x, y, z, 0)^T$

广义的齐次坐标表示: 对于点 $(x, y, z, w)(w \neq 0)$ 而言，齐次坐标的含义表示为 $(x/w, y/w, z/w)$.

3D 仿射变换的齐次坐标表示

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.14}$$

$$\boldsymbol{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.15}$$

对于三维旋转，主要考虑绕 x, y, z 轴的基本旋转变换，如下式所示，对于围绕的坐标轴坐标保持不变，其他的维度所构成的旋转变换子矩阵就是 2D 的绕轴旋转变换矩阵。

$$\boldsymbol{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R}_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.16}$$

其中，对于绕 y 轴旋转的矩阵变换 $\boldsymbol{R}_y(\alpha)$ 由于 $z \times x$ 得到 y 轴，因此对于变换 $\sin\alpha$ 额外取负。

5.3.2 3D Rotations

对于任意的 3D 旋转变换，可以将其分解为绕 x, y, z 轴的基本变换的组合，即

$$\boldsymbol{R}_{xyz}(\alpha, \beta, \gamma) = \boldsymbol{R}_x(\alpha)\boldsymbol{R}_y(\beta)\boldsymbol{R}_z(\gamma) \tag{5.17}$$

这也被称为欧拉角（Euler Angles），三个旋转动作分别称为：roll, pitch, yaw，对应的旋转轴相互垂直，就是取得 x, y, z 轴，如下图所示：

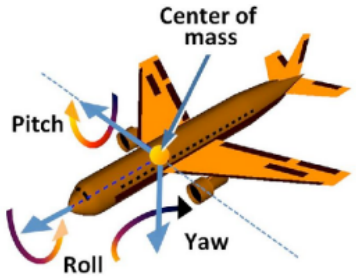


Figure 5.4: 欧拉角对应的三个旋转轴

5.3.3 Rodrigues's Rotation Formula

对于绕特定轴 $\vec{n}$ 旋转任意角度 α 计算对应的旋转变换矩阵，可以通过分解的思想：先将旋转轴平移到坐标轴上，然后绕坐标轴旋转，最后将旋转后的图形全部移到原来的旋转轴上。该公式形式化计算为，

$$\boldsymbol{R}(\vec{n}, \alpha) = \cos(\alpha)\boldsymbol{I} + (1 - \cos(\alpha))\boldsymbol{n}\boldsymbol{n}^T + \sin(\alpha) \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \tag{5.18}$$

5.4 Viewing Transformation

观测变换主要包括 View(视图)/Camera Transformation 以及 Project(投影) Transformation，其中投影变换又包括 Orthographic(正交) Projection 以及 Perspective(透视) Projection.

5.4.1 View/Camera Transformation

类比相机照相过程，Model Transformation 是寻找位置、安排人员的过程，而 View Transformation 是寻找角度、摆放相机的过程，Projection Transformation 是最后从 3D 场景中拍摄照片，即投影到 2D 平面上的过程。

相机属性定义

在视图变换中，主要是变换相加的过程，那么需要首先定义相机在三维空间中的属性：

- Position $\vec{e}$: 相机的位置坐标;
- Look-at Direction g: 相机的朝向决拍摄的方向，即 yaw 旋转;
- Up Direction t: 相加围绕拍摄方向的旋转角度，决定拍摄的视角，即 roll 旋转;

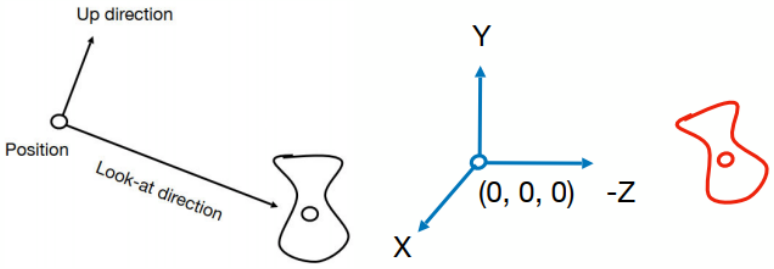


Figure 5.5: 相机的摆放方向

如图 6.1 的右图所示，相机和物体同时移动时，拍摄的内容将不保持不变，因此可以考虑将相机固定在一个特殊的位置，保持物体同时进行移动即可完成视图变换。

固定的相机位置：固定在原点，且保持 Up Direction 与 y 轴保持一致，Look-at Direction 与 $-z$ 轴保持一致。

视图变换矩阵计算

在固定相机位置的过程中，其实就是对视图进行变换的过程，固定好相机后，物体的位置也随即确定。视图变换矩阵 M_{view} 是将相机移至之前固定位置的一系列操作的组合，即

- Translates to orgin
- Rotates g to $-z$
- Rotates t to y
- Rotates $g \times t$ to x

由于先平移再旋转，即可得到 $M_{view} = R_{view}T_{view}$，Translate to origin 对应的平移矩阵可以认为将相机坐标 $\vec{e}$ 平移 $-\vec{e}$ 后得到，即

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.19}$$

而旋转 Rotate g to $-z$, t to y 以及 $g\times t$ to x 可以通过逆变换来求解，即 x to $g\times t$, y to t 以及 z to $-g$ 这里注意不仅两个坐标系的轴进行了调换，对应的变的顺序也做了调换。具体的视图变换矩阵计算如下，

$$\boldsymbol{R}_{view}^{-1}=\begin{bmatrix} x_{g\times t} & x_t & x_{-g} & 0\\ y_{g\times t} & y_t & y_{-g} & 0\\ z_{g\times t} & z_t & z_{-g} & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\Longrightarrow \boldsymbol{R}_{view}=\begin{bmatrix} x_{g\times t} & y_{g\times t} & z_{g\times t} & 0\\ x_t & y_t & z_t & 0\\ x_{-g} & y_{-g} & z_{-g} & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{5.20}$$

两点解释

1. 对于 $\boldsymbol{R}_{view}^{-1}$ 而言，可以取 xyz 坐标系下各坐标轴的单位向量，记为 $\vec{x}=[1,0,0]^T,\vec{y}=[0,1,0]^T,\vec{z}=[0,0,1]^T$，则对于旋转变换需要满足：

$$\begin{cases}\boldsymbol{R}_{view}^{-1}\cdot\vec{x}=\vec{g}\times\vec{t}\\ \boldsymbol{R}_{view}^{-1}\cdot\vec{y}=\vec{t}\\ \boldsymbol{R}_{view}^{-1}\cdot\vec{z}=-\vec{g}\end{cases} \tag{5.21}$$

   因此，根据上面的等式要求，即可得到 $\boldsymbol{R}_{view}^{-1}$.

2. 对于 $\boldsymbol{R}_{view}$ 而言，由于旋转矩阵是正交矩阵，其矩阵的逆等于矩阵的转置，因此可以得到 $\boldsymbol{R}_{view}^{T}=\boldsymbol{R}_{view}^{-1}$，进一步求出 $\boldsymbol{R}_{view}$，

$$\boldsymbol{R}_{view}=[\boldsymbol{R}_{view}^{T}]^T=[\boldsymbol{R}_{view}^{-1}]^T \tag{5.22}$$

因此，视图变换是变换相机以及关联的全部物体直到相机的位置固定在原点、向上方向与 y 轴保持一致、朝向与 $-z$ 轴保持一致。视图变换的目的是为了更好地做投影变换。

5.4.2 Projection Transformation

投影变换包括透视变换和正交变换，透视变换主要是实际观测到的图形投影，相机捕获到的光线不是平行光，造成投影过程中图形线条发生收缩现象；正交变换主要用于工程制图，是理想地对物体抽象建模，不会因为观察视角造成投影图形线条的变化。

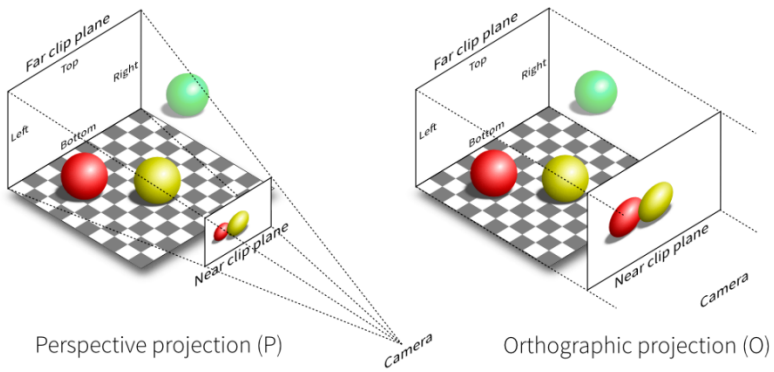


Figure 5.6: 透视投影与正交投影对比图

Orthographic Projection

正交投影/平行投影实际上将一个实际的立方体图形缩放至一个标准的立方体 $[-1,1]^3$，先将原立方体中心平移至坐标原点，然后各自对 xyz 轴的范围进行缩放，使之处于 $[-1,1]$ 之间。

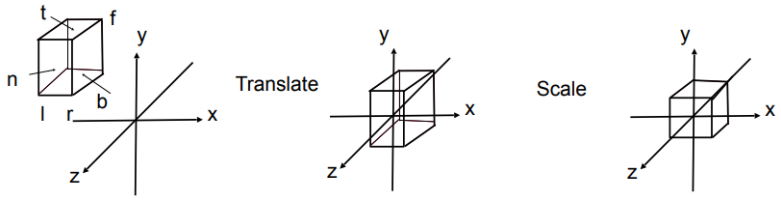


Figure 5.7: 正交投影变换

形式化的变换矩阵表示为，

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

因此，正交变换之后会对物体形状造成拉伸，都被变换到 $[-1,1]^3$ 立方体之间。

Perspective Projection

透视投影满足近大远小的性质，在图形学中非常常见。计算透视投影，可以将其转换为计算透视投影到正交投影的变换，而正交投影已经计算出。如下图所示，对于圆锥街头体（Frustum）图形，将远平面进行压缩直至变为一个立方体，并保证面中心保持不变。

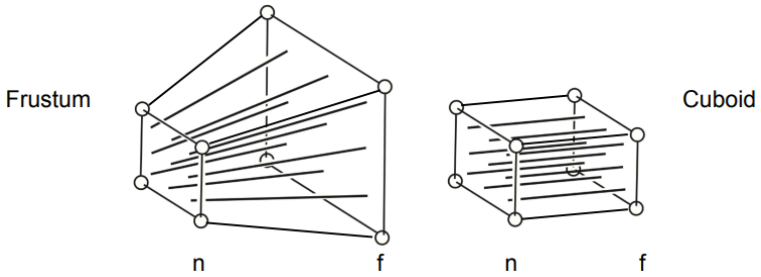


Figure 5.8: 透视投影变换为正交投影

计算对应变换前后的坐标关系，如下图对物体投影的分析所示，根据三角形相似性可以计算出远平面压缩后对应的 y 坐标为：$y' = \frac{n}{z}y$.

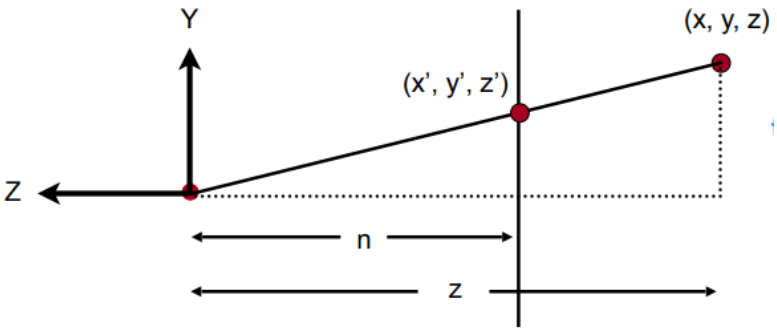


Figure 5.9: 透视投影变换为正交投影

因此可以得到，远平面变换后的坐标为，

$$y' = \frac{n}{z}y, x' = \frac{n}{z}x \tag{5.24}$$

但是由于物体压缩会导致物体上各点的分布变化，对应的 z 坐标就无法确切的知道。在齐次坐标系下，有如下关系，

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} nx/z \\ ny/z \\ unknown \\ 1 \end{bmatrix} == \begin{bmatrix} nx \\ ny \\ still\ unknown \\ z \end{bmatrix} \tag{5.25}$$

那么我们一定知道，透视到正交投影变换矩阵一定会满足下式，

$$M_{persp\to ortho}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ unkonwn \\ z \end{bmatrix} \Longrightarrow M_{persp\to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.26}$$

利用其它信息来计算剩余的矩阵元素，

1. 近平面上的点不会改变

   对于近平面上的点，用 n 替换 z 可以得到，坐标点 $[x, y, n, 1]^T$，应用 $M_{persp\to ortho}$ 得到如下公式：

   $$M_{persp\to ortho} \cdot \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ unkown \\ n \end{bmatrix} = \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} \tag{5.27}$$

   根据变换前后坐标不变的性质，可以得出 unknown 是一定是 n^2，因为 $[nx, ny, n^2, n] == [x, y, z, 1]$。因此，我们可以得到推导出的关系式，

   $$[0, 0, A, B]\begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} = n^2 \Longrightarrow An + B = n^2 \tag{5.28}$$

   其中 n^2 与 x, y 无关，所有向量的前两项一定为 0.

2. 远平面上的点 z 坐标不会改变

   对于远平面上的点，可以选取一个 z 轴上的交点 $[0, 0, f, 1]^T$ 进行计算，

   $$\begin{bmatrix} 0 \\ 0 \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ unknown \\ f \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ f \\ 1 \end{bmatrix} \Longrightarrow unkown = f^2 \Longrightarrow Af + B = f^2 \tag{5.29}$$

因此，根据以上推导出的公式可以直接求解出，$A = n + f, B = -nf$，进一步得到透视到正交的投影变换为，

$$M_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.30}$$

汇总以上结果，求出透视投影变换矩阵为，

$$M_{persp} = M_{ortho} M_{persp \to ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.31}$$

Chapter 6

Rasterization

6.1 Pixel and Screen

6.1.1 Screen

在执行 MVP (Model, View, Projection transformation) 之后，会得到一个标准的单位立方体：$[-1,1]^3$，现在就需要将立方体画在屏幕上。

屏幕（Screen）是指包含一些列像素点的二维数组，屏幕分辨率指数组的大小（长 × 宽），$1080 \times 720p$ 简称为 $720p$，$1920 \times 1080p$ 简称为 $1080p$，更高有 $2k, 4k$，因此它是一种光栅化设备。

光栅（Raster）在德语中意指“screen”，光栅化（Rasterize）即将图形画在屏幕上。

像素（Pixel）即”Picture Element” 的简称，是一个具有统一颜色的小正方块，由三原色 RGB（red, green, blue）构成。

6.1.2 Screen Space

如下图所示，对应的每一个像素单元都定义了一个对应的像素位置，默认从 0 开始计数，因此对应的位置为像素单元左下角的坐标，例如蓝色像素单元的位置为 (2, 1).

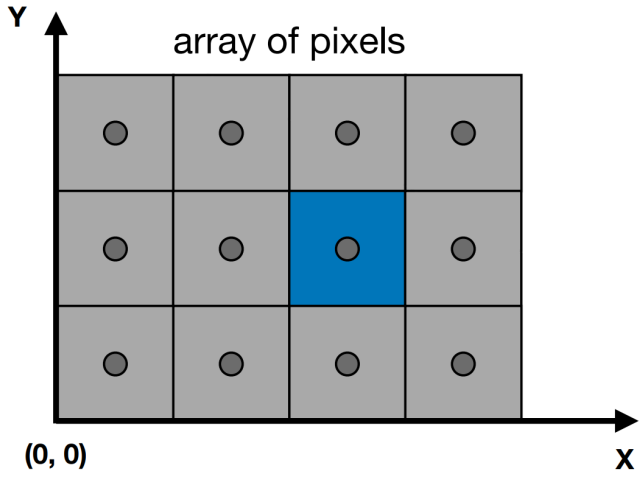


Figure 6.1: 像素空间坐标系

假设一个屏幕的分辨率为 $W \times H$，则对应的像素空间为：$[0, W-1] \times [0, H-1]$. 但是每个像素的中心都在 +0.5 偏移位置上，即像素 (x, y) 对应的像素中心位置为 $(x+0.5, y+0.5)$. 对应屏幕在像素空间中覆盖的范围为，$[0, W] \times [0, H]$.

6.1.3 Viewport Transformation

通过以上章节定义了屏幕的像素空间，现在需要将经过 MVP 得到的立方体变换到屏幕空间上，即做视图变换。

暂时不考虑 z 轴，在 $x \times y$ 平面上，将 $[-1, 1]^2$ 转换到 $[0, W] \times [0, H]$，则视图变换矩阵为：

$$M_{viewport} = \begin{bmatrix} \frac{W}{2} & 0 & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & 0 & \frac{H}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

6.2 Triangles Rasterization

将图形变换到定义的像素屏幕空间后，对应的图形还是空间坐标系中定义的矢量图形，不能称之为图像。现在就需要将这些图形离散化到对应的像素网格中，这个过程就叫做光栅化，意指将图形转换为图像，显示在屏幕上。

6.2.1 Fundamental Shape Primitives

三角形是最基本的形状原语，作为最基本的多边形，能够构成其他任意多边形，具有独特的性质：

1. 三角形内部一定是平面的；
2. 三角形内外定义是非常明确的，不会受到凹凸性的干扰，并且可以利用叉积进行判断；
3. 在三角形的顶点上进行插值可以得到属性渐变的过程（重心插值）；

6.2.2 Sampling

采样是指将一个连续函数进行离散化得到一系列函数值的过程。对应于图形学中，是采样某个屏幕像素空间中的一部分，然后对于这样采样点进行相应处理，例如判断是否是在图形内部，以此将其转换为对应的像素点。

定义二值函数 $inside(tri, x, y)$ 如下所示：

$$inside(t, x, y) = \begin{cases} 1, & \text{if Point}(x, y) \text{ is in triangle } t, \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

采样区域可以认为是整个屏幕空间，对应每一个像素点都进行 $inside(tri, x, y)$ 判断，因此可以归纳为：Rasterization = Sampling A 2D Indicator Function. 见下述代码，

```
for (int x = 0; x < xmax; ++x)
    for (int y = 0;y < ymax; ++y)
        image[x][y] = inside(tri, x + 0.5, y + 0.5);
```

回顾：叉积判定点在三角形内外

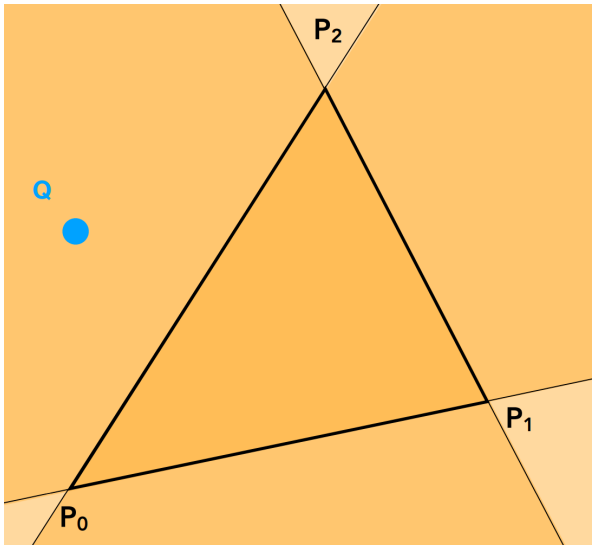


Figure 6.2: 三角形叉积判定内外

对于视图坐标系而言，z 轴指向纸面向外，判定点 Q 在 P_1P_2 的左右时，可以计算叉积 $\vec{P_1P_2} \times \vec{P_1Q}$，其中左右关系是相对于纸面面向相机的左右（默认相机是沿 $-z$ 方向）。

顺时针/逆时针标定三条边构成的向量后，以此判断 Q 点在三条边的左右关系，如果都在边的左侧/右侧，则 Q 点在三角形内部；否则在三角形外部。

6.2.3 Bounding Box Rasterization

根据上面光栅化过程，遍历整个屏幕空间判断像素点是否在三角形内部，将会额外遍历多个无关的像素点。一种解决的方式是，对于三角形添加一个边界框（如下图所示），分别取三个顶点两个维度的最大值和最小值，构成一个包围盒，随后在此包围盒中遍历像素点，将会降低一部分复杂度。

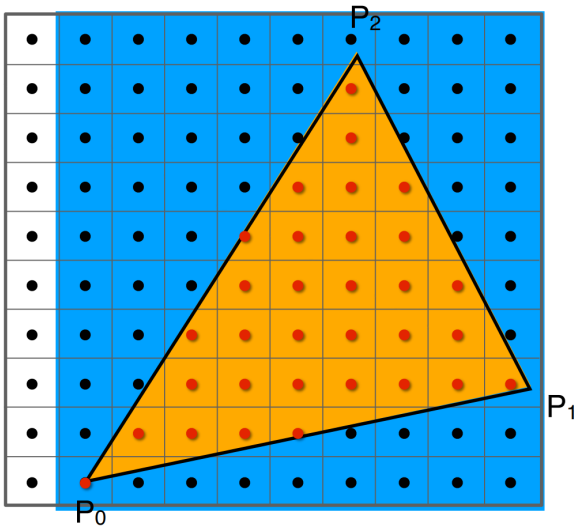


Figure 6.3: 三角形包围盒

6.3 Antialiasing and Z-Buffering

6.3.1 Antialiasing

当采样不恰当时，对应光栅化后的图形就会出现锯齿，锯齿的学名是走样（Aliasing）。这在计算机图形学中又被称为采样瑕疵 Sampling Artifacts (Errors/Mistakes/Inaccuracies)，具体体现有以下若干方面：

1. 锯齿 Jaggies，在像素空间中欠采样；
2. 摩尔纹 Moire，在图像中欠采样；
3. 齿轮效应 Wagon wheel effect，在时间上欠采样；

信号频率较高，而采样频率较低，出现欠采样现象。

Frequency Domain

反走样：对信号做模糊（滤波），再做采样，