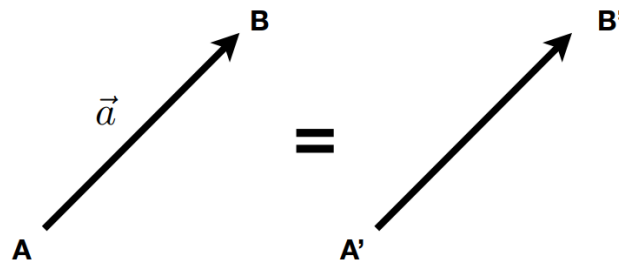


Graphics Notes

The Quick Brown Fox Jumps Over The Lazy Dog
道理我都懂，鸽子为什么那么大

Linear Algebra

Vectors



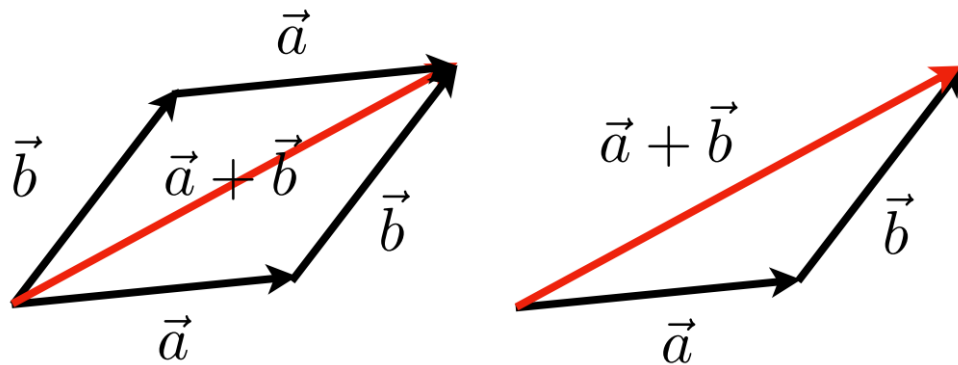
- Usually written as \vec{a} or in bold \mathbf{a}
- Or using start and end points $\overrightarrow{AB} = B - A$
- Direction and length
- No absolute starting position

Vector Normalization

- Magnitude (length) of a vector written as $\|\vec{a}\|$
- Unit vector
 - A vector with magnitude of 1
 - Finding the unit vector of a vector (normalization): $\hat{a} = \vec{a} / \|\vec{a}\|$
 - Used to represent directions

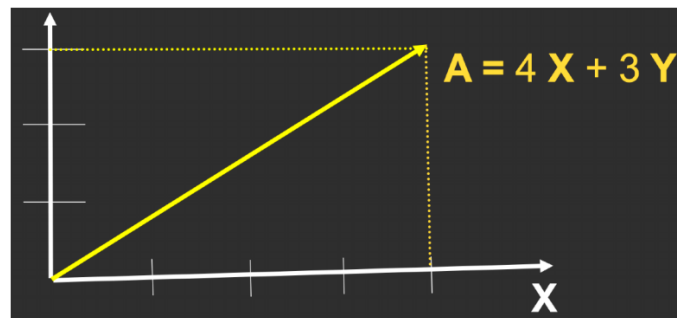
单位向量一般读作 aHat (a 上面的小尖角像是一顶帽子)

Vector Addition



- Geometrically: Parallelogram law & Triangle law
- Algebraically: Simply add coordinates

Cartesian Coordinates



- X and Y can be any (usually **orthogonal unit**) vectors

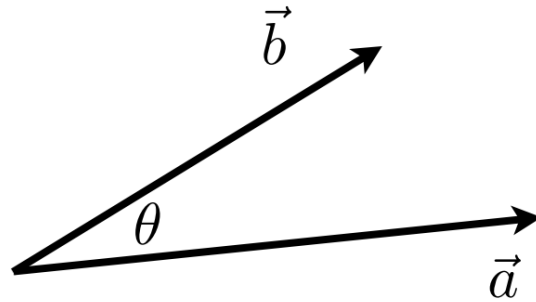
$$\mathbf{A} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \mathbf{A}^T = (x, y) \quad \|\mathbf{A}\| = \sqrt{x^2 + y^2}$$

如果不做特别说明的话，默认是列向量（竖着写），转置（行列互换）后为横向量，使用列向量矩阵可以左乘

Vector Multiplication

Dot product

用于求两向量之间的夹角（两个向量点乘，结果是一个标量）



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

- For unit vectors

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

$$\cos \theta = \hat{a} \cdot \hat{b}$$

点乘的性质满足交换律，结合律和分配律：

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

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

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

在笛卡尔坐标系中，先将对应部分相乘，然后相加

- In 2D

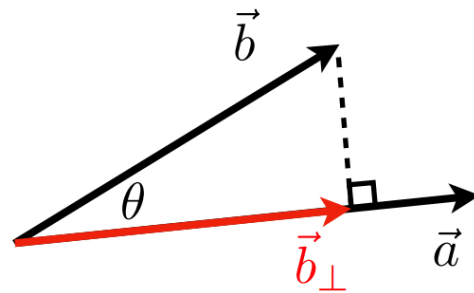
$$\vec{a} \cdot \vec{b} = \begin{pmatrix} x_a \\ y_a \end{pmatrix} \cdot \begin{pmatrix} x_b \\ y_b \end{pmatrix} = x_a x_b + y_a y_b$$

- In 3D

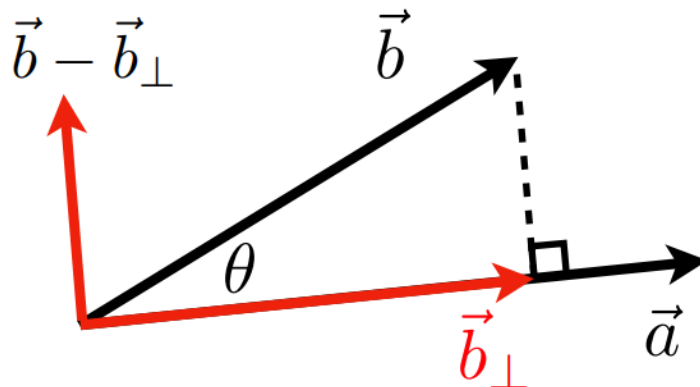
$$\vec{a} \cdot \vec{b} = \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} \cdot \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = x_a x_b + y_a y_b + z_a z_b$$

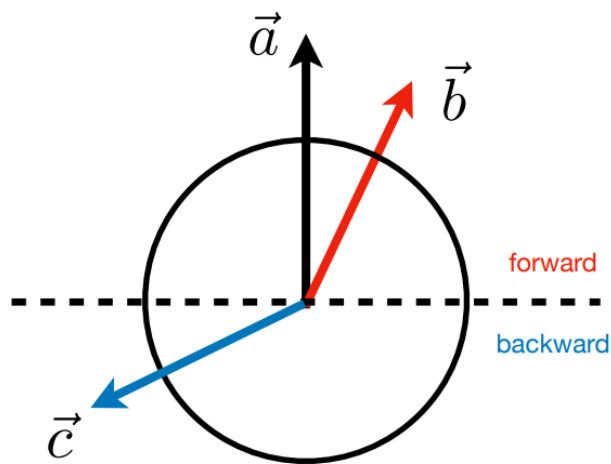
用于计算一个向量在另一个向量上的投影，投影算出来后还可以把这个向量进行平行与垂直方向的分解：

- \vec{b}_{\perp} : projection of \vec{b} onto \vec{a}
 - \vec{b}_{\perp} must be along \vec{a} (or along \hat{a})
 - $\vec{b}_{\perp} = k\hat{a}$
 - What's its magnitude k?
 - $k = \|\vec{b}_{\perp}\| = \|\vec{b}\| \cos \theta$



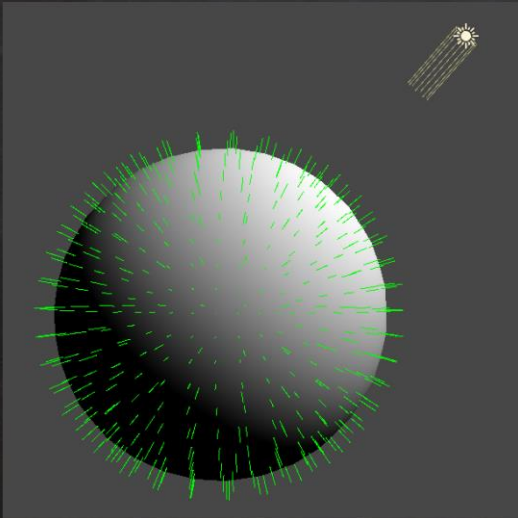
向量 \mathbf{b} 在向量 \mathbf{a} 上的投影读作 \mathbf{b} Perp(perpendicular)





dot product > or < 0

用两个向量点乘的结果与 0 比较来判断方向，如果大于 0 则说明方向接近一致，等于零说明互相垂直，小于零则方向接近相反，下面是一些光照模型中的应用（nDotl, rDotv, ndoth）



根据向量点乘的图形学含义，我们令：

- 模型表面的垂直方向为向量nDir(即：法线方向)；
- 令光照方向的反方向为向量lDir(即：LightDir)；
- 令nDir · lDir(两者点乘)结果为像素输出；

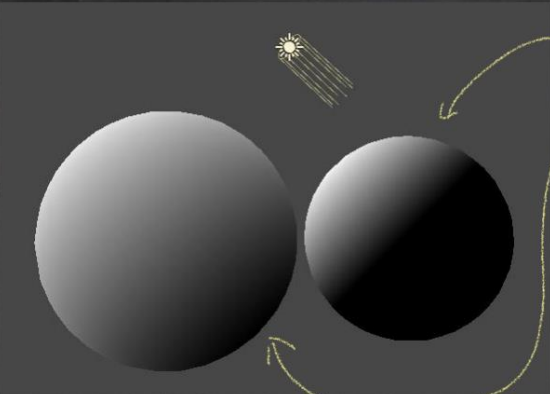
则有如左图的光照表现，其中：

- 最亮处：值为1，纯白；
- 明暗交界处：值为0，纯黑；
- 暗部：值为负数，亦为黑；

负数是个无意义的亮度，所以我们通常把结果为负数的值，都改为0：

- 则有：Max(0, nDir · lDir)；
- 即为兰伯特光照模型；

Lambert



我们回到上页这个状态：

如果对其明度乘以0.5，再加上0.5，则有

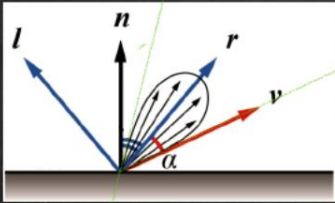
这种光照模型叫：

- 半Lambert光照模型；
- 优点：比Lambert透气，不至于暗部死黑；
- 缺点：显然和素描老师教的不一样；

Half-Lambert

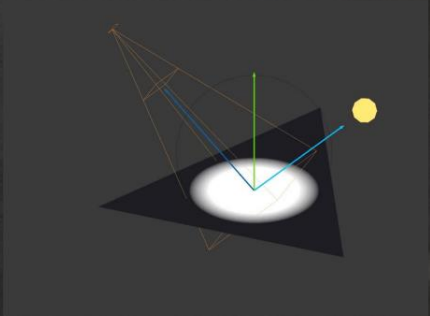
04 镜面反射

可观察 Specular 的视角范围



镜面反射-Specular:

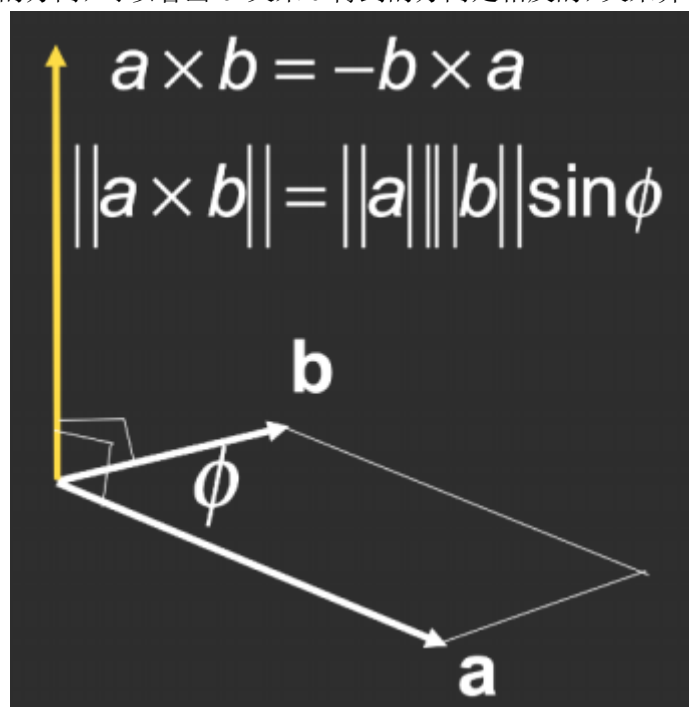
- 因其反射有明显方向性，所以观察者的视角决定了反射光线的有无，明暗；
- 实现方式：
 - Phong ($r \cdot v$)，即光反射方向和视角方向越重合，反射越强；
 - Blinn-Phong ($n \cdot h$)，即法线方向和半角方向越重合，反射越强；



Phong and Blinn-Phong

Cross product

两个向量叉乘，结果也是一个向量，且结果的向量垂直于叉乘的两个向量组成的平面，叉乘结果的方向可以用右手螺旋定则判断（ a 叉乘 b ，手掌伸直，四指由 a 握向 b ，叉乘结果的方向为大拇指的方向，可以看出 b 叉乘 a 得到的方向是相反的，叉乘并不满足交换律）



叉乘的性质：

$$\vec{x} \times \vec{y} = +\vec{z}$$

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$$

$$\vec{y} \times \vec{x} = -\vec{z}$$

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

$$\vec{y} \times \vec{z} = +\vec{x}$$

$$\vec{z} \times \vec{y} = -\vec{x}$$

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

$$\vec{z} \times \vec{x} = +\vec{y}$$

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

$$\vec{x} \times \vec{z} = -\vec{y}$$

在笛卡尔坐标系中的计算规则：

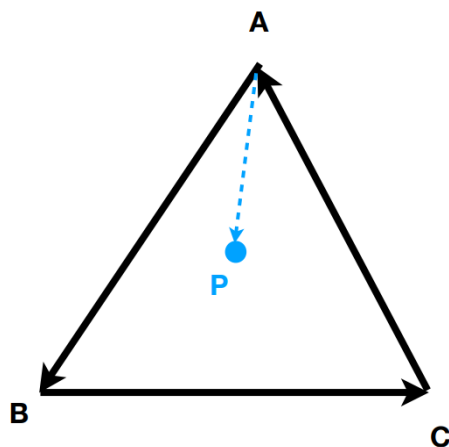
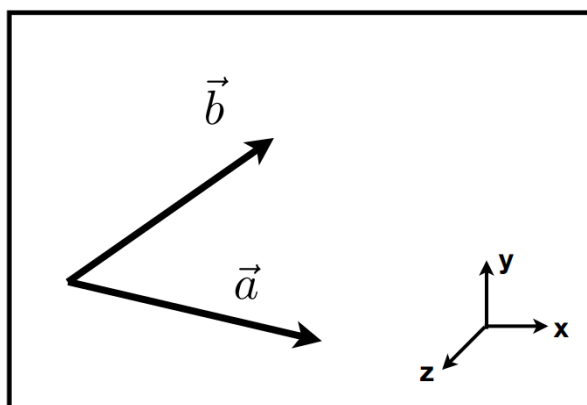
$$\vec{a} \times \vec{b} = \begin{pmatrix} y_a z_b - y_b z_a \\ z_a x_b - x_a z_b \\ x_a y_b - y_a x_b \end{pmatrix}$$

• Later in this lecture

$$\vec{a} \times \vec{b} = A^* b = \begin{pmatrix} 0 & -z_a & y_a \\ z_a & 0 & -x_a \\ -y_a & x_a & 0 \end{pmatrix} \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix}$$

dual matrix of vector a

用于判断两个向量间的左右关系，点在多边形的内外关系（判断在多边形内外时会出现叉乘为 0 向量，点正好在边上的情况 Corner Case，这时一般由自己决定边界情况算是内还是外）；判断点 P 是否在三角形 ABC 内可以通过计算 AB AP, BC BP, CA CP 的叉乘结果是否同大于（小于）0 来得出：



Orthonormal bases and coordinate frames

三个向量同时满足以下条件时即可构成一个右手坐标系：

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

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

$$\vec{w} = \vec{u} \times \vec{v} \quad (\text{right-handed})$$

并且可以将投影进行分解：

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

(projection)

Matrices

Array of numbers ($m \times n = m$ rows, n columns):

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix}$$

Matrix-Matrix Multiplication

两个 **AB** 矩阵相乘的规则是 **A** 矩阵的行数要等于 **B** 矩阵的列数，相乘的结果是一个有着 **A** 矩阵行数，**B** 矩阵列数的新矩阵，例如一个 **M** 行 **N** 列矩阵 **A**，与一个 **N** 行 **P** 列的矩阵相乘，得到的是一个 **M** 行 **P** 列的矩阵

这边闫老师提出了一个很好的计算方式，结果矩阵的第 *i* 行 *j* 列的数可以由 **A** 矩阵的第

i 行点乘 B 矩阵的第 j 行得来，如下图第 1 行第 2 列的？，计算结果为 A 矩阵的第 1 行点乘 B 矩阵的第 2 列，也就是 $1*3+3*2=9$ ；第 3 行第 4 列的？，计算结果为 A 矩阵的第 3 行点乘 B 矩阵的第 4 列，也就是 $0*4+4*3=12$

- # (number of) columns in A must = # rows in B
 $(M \times N) (N \times P) = (M \times P)$

$$\begin{pmatrix} 1 & 3 \\ 5 & 2 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} 3 & 6 & 9 & 4 \\ 2 & 7 & 8 & 3 \end{pmatrix} = \begin{pmatrix} 9 & ? & 33 & 13 \\ 19 & 44 & 61 & 26 \\ 8 & 28 & 32 & ? \end{pmatrix}$$

- Element (i, j) in the product is
the dot product of row i from A and column j from B

性质：大部分不满足交换律（只有一些特殊的矩阵在交换顺序相乘时结果也相同），满足结合律和分配律：

– **Non-commutative**
(AB and BA are different in general)

- Associative and distributive
- $(AB)C = A(BC)$
 - $A(B+C) = AB + AC$
 - $(A+B)C = AC + BC$

Matrix-Vector Multiplication

把向量看成是一个 m 行一列的矩阵，一个向量乘矩阵可以完成旋转、平移、缩放等操作，下面是一个二位向量乘矩阵得到与 y 轴对称的向量：

- Treat vector as a column matrix ($m \times 1$)
- Key for transforming points (next lecture)

- Official spoiler: 2D reflection about y-axis

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix}$$

Transpose of a Matrix

矩阵转置：矩阵的行列互换（将矩阵顺时针转 90° ）：

- Switch rows and columns (ij \rightarrow ji)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

性质：穿脱原则，类似于栈，先入后出

$$(AB)^T = B^T A^T$$

Identity Matrix and Inverses

只有对角线上有非 0 元素的是单位矩阵

如果一个矩阵 A 乘另一个矩阵得到的结果是单位矩阵，那么这两个矩阵是互逆的
与矩阵的转置性质相同，穿脱原则，类似于栈，先入后出

$$I_{3 \times 3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$AA^{-1} = A^{-1}A = I$$

$$(AB)^{-1} = B^{-1}A^{-1}$$

Vector multiplication in Matrix form

点乘和叉乘都可以写成矩阵的形式：

- Dot product?

$$\begin{aligned}\vec{a} \cdot \vec{b} &= \vec{a}^T \vec{b} \\ &= (x_a \quad y_a \quad z_a) \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = (x_a x_b + y_a y_b + z_a z_b)\end{aligned}$$

- Cross product?

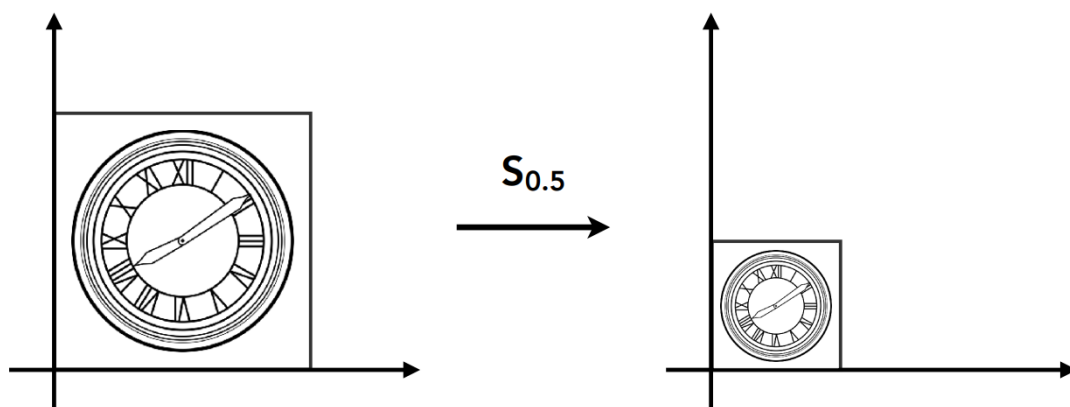
$$\vec{a} \times \vec{b} = A^* b = \begin{pmatrix} 0 & -z_a & y_a \\ z_a & 0 & -x_a \\ -y_a & x_a & 0 \end{pmatrix} \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix}$$

dual matrix of vector a

Transformation

在光栅化成像的过程中大量用到了变换，常见的 2 维空间的变换有旋转，缩放和切变：
2D transformations: rotation, scale, shear

Scale



这边由左图缩放到右图，假设左图的坐标为 x, y ，变换后的右图为 $x_{\text{Prime}}, y_{\text{Prime}}$ ，缩放的比例为 s ，则有：

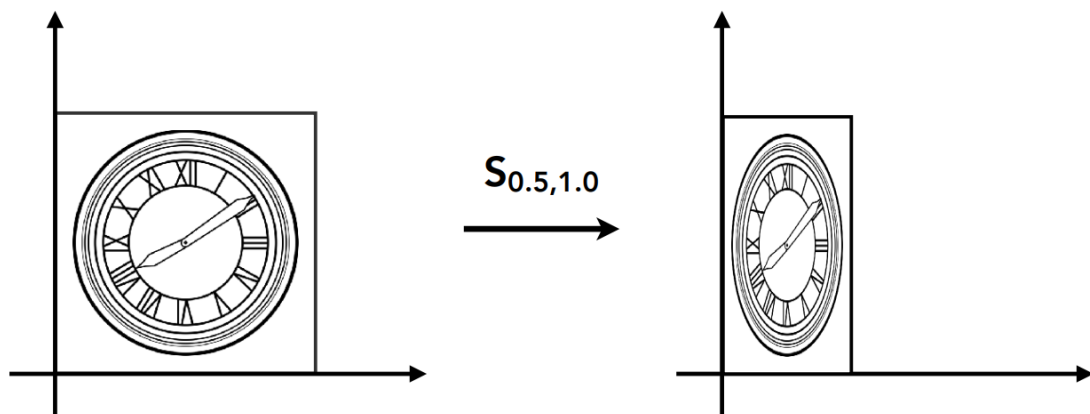
$$x' = sx$$

$$y' = sy$$

写成矩阵的形式，可以得到缩放矩阵：

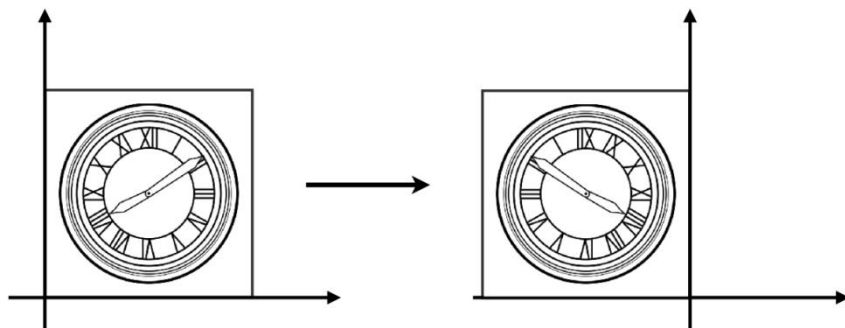
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

xy 轴的缩放比例各不相同也是同理，仍然可以写成这样的矩阵：



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

将原本的图片做一个对 y 轴的反射操作（镜像），可以看出翻转的 y 坐标不变，x 坐标相反，可以写出变换的矩阵：

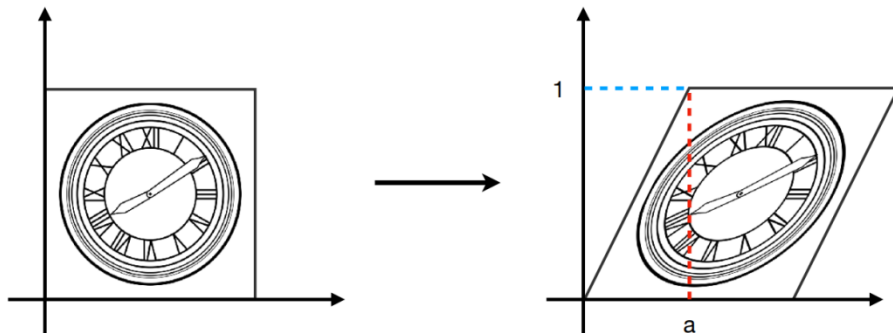


Horizontal reflection:

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned}$$

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

将左边的图看作是有弹性可以拉伸的，形成右侧的图形，将之称为切变（Shear），可以看出 y 坐标是不变的，水平方向变换之后可以用 $x+ay$ 来表示，即可算出对应的切变矩阵（下面附上推算过程）



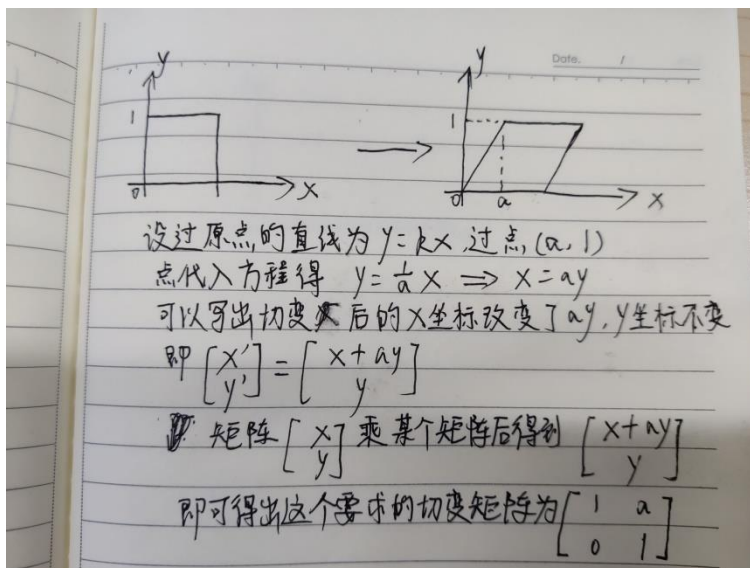
Hints:

Horizontal shift is 0 at $y=0$

Horizontal shift is a at $y=1$

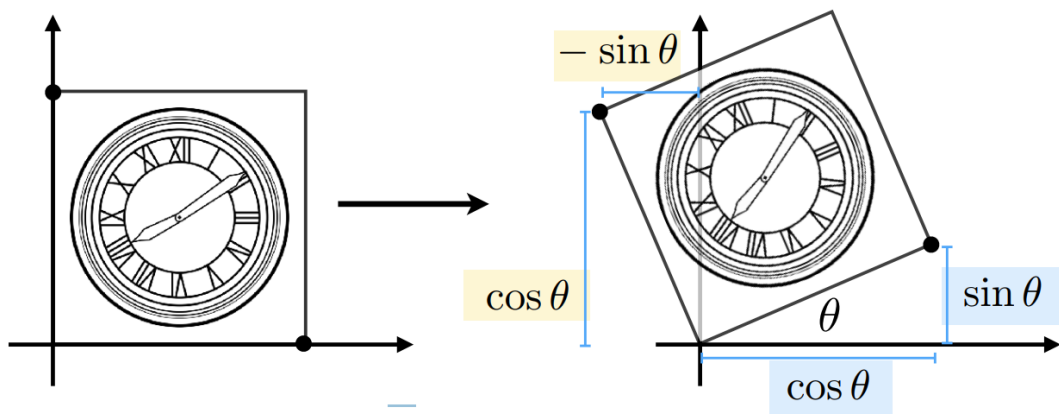
Vertical shift is always 0

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



Rotate (about the origin (0, 0), CCW by default)

不做特别说明的话默认围绕 $(0, 0)$ 点逆时针旋转：下面附上闫大神的矩阵推导与自己的另一种矩阵推导方式



闫大神通过两个特殊的点推导出来（特殊值法 yyds）：

$(x, y) \approx (x', y')$
 $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$
 $\begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
 $\cos \theta = A \cdot 1 + B \cdot 0 = A$
 $\sin \theta = C \cdot 1 + D \cdot 0 = C$

通过三角函数公式推导出来：

设点 (x, y) 逆时针绕原点旋转 θ 角度，原点到 (x, y) 长度为 r
 可得 $x = r \cos \alpha$ $y = r \sin \alpha$
 ~~$x = r \cos \alpha$~~
 $x' = r \cos(\alpha + \theta)$ $y' = r \sin(\alpha + \theta)$
 $= r \cos \alpha \cos \theta - r \sin \alpha \sin \theta$ $= r \sin \alpha \cos \theta + r \cos \alpha \sin \theta$
 代入 $= x \cos \theta - y \sin \theta$ $= y \cos \theta + x \sin \theta$
 $\therefore \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ y \cos \theta + x \sin \theta \end{bmatrix}$
 可得出旋转矩阵
 $\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$

可以用矩阵相乘输入的点得到变换后的点，这种变换叫做线性变换

Linear Transforms = Matrices

(of the same dimension)

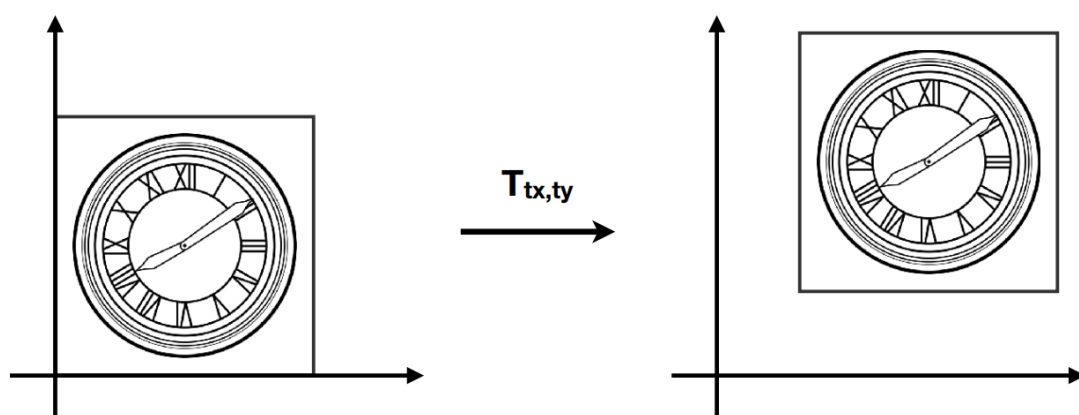
$$x' = a x + b y$$

$$y' = c x + d y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{M} \mathbf{x}$$

旋转和缩放变换都可以用相同维度的矩阵相乘来表示，平移变换为了便于用矩阵相乘表示引入其次坐标的概念，如下我们把左图平移 t_x, t_y ，平移后得到 $x_{\text{prime}}, y_{\text{prime}}$



$$x' = x + t_x$$

$$y' = y + t_y$$

但是想要把变换后的图形写成矩阵相乘的形式（维度不变），就只能写成下面这样，先进行线性变换，然后再平移：

- Translation cannot be represented in matrix form

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

(So, translation is NOT linear transform!)

- But we don't want translation to be a special case
- Is there a unified way to represent all transformations?
(and what's the cost?)

由于不想把平移变换当成特例来处理，那么有没有一个方法能够统一表示所有的变换，统一表示的代价如何（tradeOff）有人提出了解决方法，引入齐次坐标，增加一个维度，同时代价是数据量增加了：

Add a third coordinate (w-coordinate)

- 2D point = (x, y, 1)^T
- 2D vector = (x, y, 0)^T

Matrix representation of translations

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

为什么表示点引入的 w 坐标为 1，而表示向量引入的 w 坐标为 0？因为这样表示恰好能够表示向量与点的一些性质，向量加向量结果是向量，点减去另一个点结果是向量，点加向量结果是点而点与点相加其实是没什么的意义的（1+1=2），在齐次坐标中的意义是两个点的中点：

Valid operation if w-coordinate of result is 1 or 0

- vector + vector = vector
- point - point = vector
- point + vector = point
- point + point = ??

In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} \text{ is the 2D point } \begin{pmatrix} x/w \\ y/w \\ 1 \end{pmatrix}, w \neq 0$$

三维变换中也是同理：

Use homogeneous coordinates again:

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

In general, (x, y, z, w) ($w \neq 0$) is the 3D point:

$$(x/w, y/w, z/w)$$

Affine Transformations

Affine map = linear map + translation

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

Using homogenous coordinates:

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

三维空间中的仿射变换也是如此，增加一个维度使用 4×4 的矩阵：

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

在表示二维下的仿射变换矩阵时，最后一行是 0, 0, 1；以下是一些二维变换的矩阵：

Scale

$$\mathbf{S}(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$\mathbf{R}(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

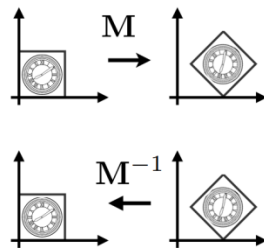
Translation

$$\mathbf{T}(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Inverse Transform

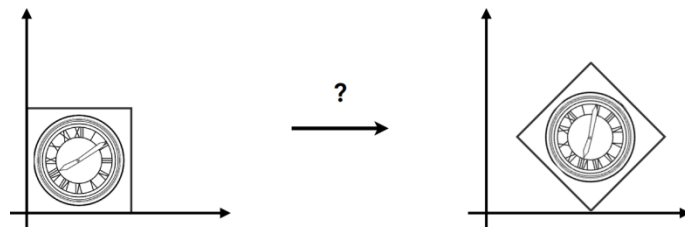
逆矩阵用于表示一个图形的逆变换，假如输入图形 **A** 乘矩阵 **M** 变换后得到矩阵 **B**,那么矩阵 **B** 乘矩阵 **M** 的逆矩阵后还原到变换前的矩阵 **A**(做了一个操作之后再做一个反的操作，相当于什么操作都没做)

\mathbf{M}^{-1} is the inverse of transform \mathbf{M} in both a matrix and geometric sense

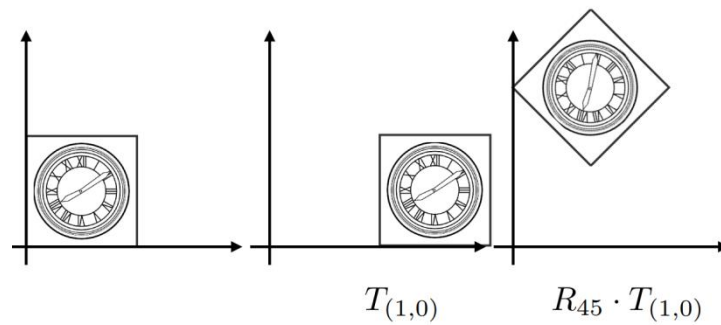


Composing Transforms

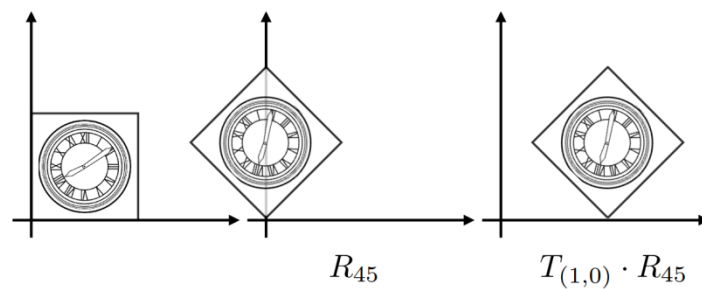
复杂的变换可以通过简单的变换组合而成，在过程中可以发现变换的顺序是不能改变的，如下图，将左边的图形变换到右边的图形，先平移后旋转与先旋转后平移得到的结果不相同：



先平移后旋转：



先旋转后平移：



印证了矩阵乘法是不满足交换律的

Matrix multiplication is not commutative

$$R_{45} \cdot T_{(1,0)} \neq T_{(1,0)} \cdot R_{45}$$

Note that matrices are applied right to left:

$$T_{(1,0)} \cdot R_{45} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

同时组合变换可以使用矩阵乘法的结合律，可以用一个组合变换的矩阵来表示复杂的变换，这是一个非常好用的性质：

Sequence of affine transforms A_1, A_2, A_3, \dots

- Compose by matrix multiplication
- Very important for performance!

$$A_n(\dots A_2(A_1(\mathbf{x}))) = \underbrace{A_n \cdots A_2 \cdot A_1}_{\text{Pre-multiply } n \text{ matrices to obtain a single matrix representing combined transform}} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

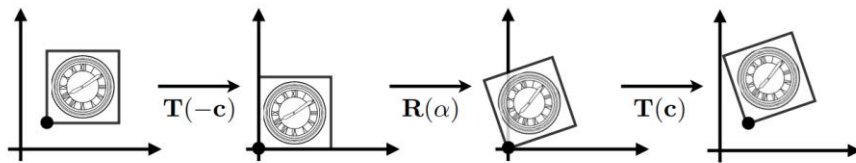
Pre-multiply n matrices to obtain a single matrix representing combined transform :

Decomposing Complex Transforms

一般不做特别说明的话，做旋转是围绕（0，0）点进行旋转的，如果想围绕任意一点 c 进行旋转，我们先把图形平移 $-c$ 到原点的位置，旋转后再平移 c 移回去，写成矩阵的话要注意矩阵乘法是从右到左进行应用的：

How to rotate around a given point c ?

1. Translate center to origin
2. Rotate
3. Translate back



Matrix representation?

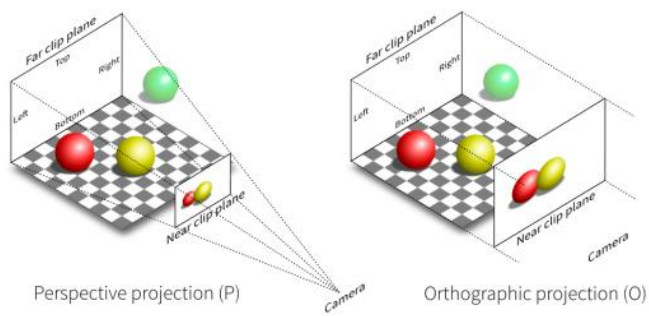
$$T(c) \cdot R(\alpha) \cdot T(-c)$$

SS

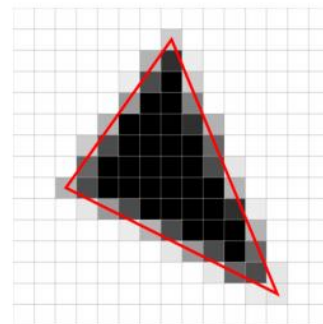
Rasterization

Overview

- 把三维空间中的几何形体显示到屏幕中
光栅化是实时的计算机图形学的主要应用，在计算机图形学中将实时定义为每秒能够生成 30 幅画面，也叫 30 帧（30fps），如果不能达到的话则叫做离线
- 将原物体投影映射到像素
- 游戏的黄金标准



<http://vispy.org/modern-gl.html>



https://commons.wikimedia.org/wiki/File:Rasterisation-triangle_example.svg

光栅化过程中三角形遍历的时候可以应用叉乘来判断像素的中心点是否在三角形内，例如图 1 中的红点（在三角形内）和白点（在三角形外），

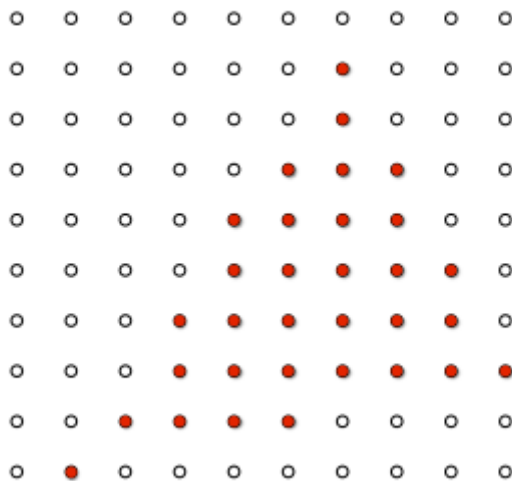


图 1

然后把对应的像素进行着色，就会得到如图 2 这样的类似三角形的图案，有很多的锯齿，和期望得到的图 3

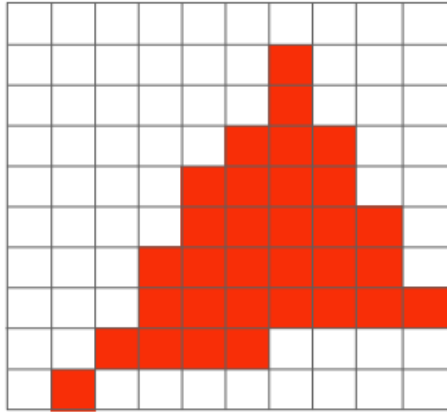


图 2

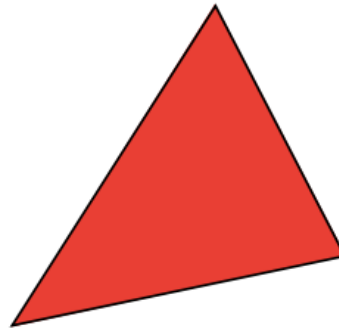
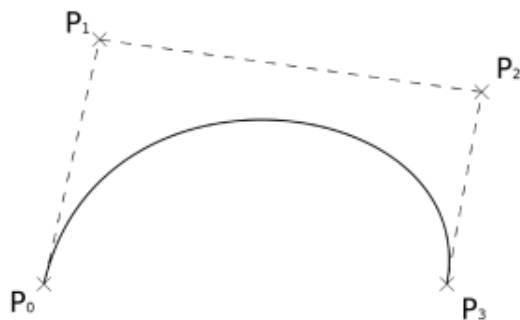


图 3

Curves and Meshes

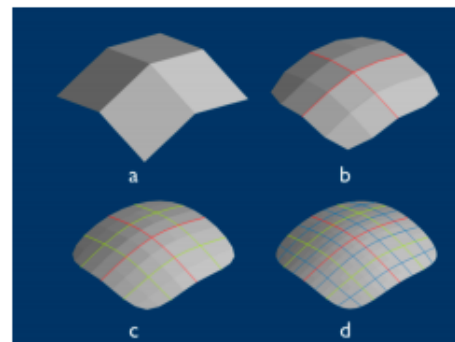
Overview

如何表示一条光滑的曲线、如何表示曲面、如何用简单的曲面通过细分来得到一些更复杂的曲面，在形状发生变化时这些面要如何变化，如何保持住物体的拓扑结构



Bezier Curve

https://en.wikipedia.org/wiki/Bezier_curve



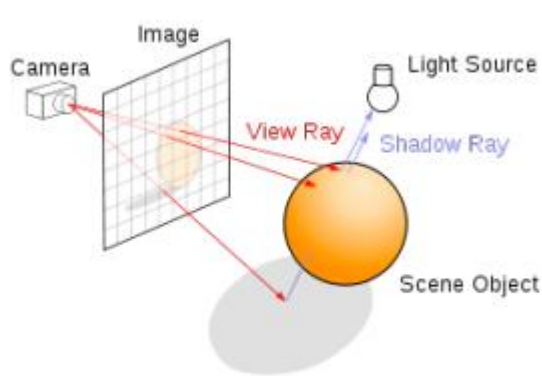
Catmull-Clark subdivision

https://commons.wikimedia.org/wiki/File:Catmull-Clark_subdivision_of_4_planes.png

Ray Tracing

Overview

图形学乃至是生活中有很多事情其实可以看作 **tradeoff**，在为了达成某一个目标不得不牺牲一些其他的目标，有一个取舍与权衡的过程，光线追踪能够生产非常真实的画面，如下右图，但是速度慢，在动画和电影中使用较多。



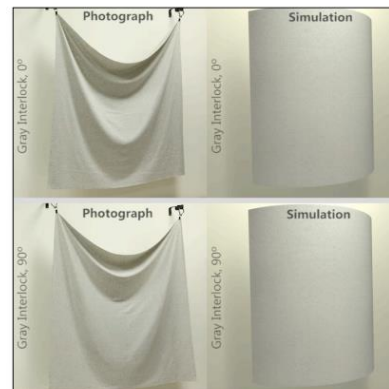
[https://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))

Animation / Simulation

Overview

动画与模拟仿真，一个弹性小球落在地上，与地面进行挤压发生形变弹上去再下来，布料自然下垂等

- Key frame Animation
- Mass-spring System



https://cs184.eecs.berkeley.edu/sp18/lecture/simulation/slide_010