table of contents

Contents

1 Introduction

2 Misc Math

1. **(trig identities)**
2. **(dot product)**
$$\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\cos\phi \qquad proj_{\mathbf{a}}(\mathbf{b}) = \|\mathbf{a}\|\cos(\phi) = \frac{a\cdot b}{\|\mathbf{b}\|}$$
3. **(cross product)**
$$\mathbf{a}\times\mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\sin(\phi)\mathbf{n} = \det\begin{pmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k}\\ a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\end{pmatrix}$$
where $\mathbf{n}$ is unit vector perpendicular to plane formed by $\mathbf{a},\mathbf{b}$, determined by the right hand rule
4. **(coordinate frames)**
   - the **global coordinate system** is one formed by cartesian canonical orthonormal basis and the canonical origin that is not stored explicitly.
   - Other coordinate system is called a **frame of reference** or **coordinate frame** are stored explicitly (the origin, and 3 orthonormal vectors as a function of the canonical basis) and is called a **local coordinate system**
   - If $\mathbf{b}$ is in canonical $\mathbf{x}-\mathbf{y}-\mathbf{z}$ coordinate and want to transform to a local $\mathbf{u}-\mathbf{v}-\mathbf{w}$ coordinate. then the transformed vecter can be written as
$$\sum_{\mathbf{q}\in\{\mathbf{u},\mathbf{v},\mathbf{w}\}} \langle\mathbf{q},\mathbf{b}\rangle\,\mathbf{q}$$
5. **(construct basis)** by GramSchmidt process
6. **(linear interpolation)** over 2 points or a set of points where the function is piece-wise linear
7. **(triangles)**
   - barycentric coordinates is a nonorthogonal makes interpolation straight-forward over a triangle. Say $(\mathbf{a},\mathbf{b},\mathbf{c})$ forms a triangle, then the vertices $(\mathbf{a},\mathbf{b},\mathbf{c})$ act as basis vectors and form the barycentric coordinates. Any point $\mathbf{p}$ is
$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \qquad \alpha+\beta+\gamma = 1$$
(α,β,γ) is the barycentric coordinate of $\mathbf{p}$ w.r.t. $\mathbf{a},\mathbf{b},\mathbf{c}$. Given $\mathbf{p}$ in cartesian coordinate, we can find its baarycentric coordinate by solving
$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \qquad \Longleftrightarrow \qquad \begin{pmatrix}x_b - x_a & x_c - x_a\\ y_b - y_a & y_c - y_a\end{pmatrix}\begin{pmatrix}\beta\\ \gamma\end{pmatrix} = \begin{pmatrix}x_p - x_a\\ y_p - y_a\end{pmatrix}$$
Geometrically, α,β,γ can be computed using triangle area
$$\alpha = A_\alpha/A \quad \beta = A_\beta/A \quad \gamma = A_\gamma/A$$
Properties
     - can detect pointson edge/verties easily
     - mixes coordinates of vertices smoothly
     - can detect insidedness of points easily, i.e. $\alpha,\beta,\gamma > 0$

3 Raster Images

1. **(raster display)** show images as rectangular arrays of pixels.
2. **(raster images)** a 2D array that stores pixel value for each pixel that is device-independent
3. **(vector images)** description of shapes that is resolution independent but needs to be rasterized before display
4. **(point sample)** the reflectance, or fraction of light reflected as a function of position on a piece of paper

   $$I(x, y) : R \subset \mathbb{R}^2 \to V$$

   where V is the set of possible pixel values, i.e. $V = \mathbb{R}^+$ for grayscale images. Pixel in raster images are local average of the color of the image, called a point sample of the image
5. **(pixel values)** as tradeoff between memory and resolution. Lowered bits introduce artifacts such as clipping and banding
6. **(RGB color)**
   (a) (pixel coverage) α is the fraction of pixerl covered by the foreground layer.
   (b) (composite pixel) Let $\mathbf{c}_f, \mathbf{c}_b$ be foreground and background colors respectively, then $\mathbf{c} = \alpha\mathbf{c}_f + (1 - \alpha)\mathbf{c}_b$
   (c) **(alpha/transparency mask)** α values for all pixels, stored as a separate grayscale image; or stored as a 4th channel, called the alpha channel

4 Ray Tracing

1. **(rendering)** take a scene, a model, composed of many geometric objects arranged in 3D space and produce a 2D image that shows the objects as viewed from a particular viewpoint
   - **(object-order rendering)** for each object, update pixels that the object influences
   - **(image-order rendering)** for each pixel, set pixels based on the objects that influence it
2. **(ray-tracing)** image-order algorithm for making renderings of 3D scenes
   (a) **(ray generation)** compute origin and direction of each pixel's viewing ray based on the camera geometry
   (b) **(ray intersection)** finds closest object intersecting the viewing ray
   (c) **(shading)** computes the pixel color based on the reseults of ray intersection
3. **(linear perspective)** 3D objects projected to image plane in such a way that stright lines in the scene become straight lines in the image
4. **(parallel projection)** 3D points are mapped to 2D by moving them along a projection direction until they hit the image plane
   - **(orthographic/oblique projection)** if image plane is perpendicular to view direction, the projection is orthogrpahic; otherwise, it is called oblique
5. **(camera frame)** let $\mathbf{e}$ be the viewpoint, and $\mathbf{u}, \mathbf{v}, \mathbf{w}$ be the three basis vectors. $\mathbf{u}$ points upward (from camera's view), $\mathbf{v}$ points upward, and $\mathbf{w}$ points backward. $-\mathbf{w}$ is called the view direction.
6. **(computing viewing ray)** represent ray with 3D parametric line

   $$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

   idea is to find $\mathbf{e}, \mathbf{d}$
   (a) **(orthographic view)** where $\mathbf{e}$ the viewpoint is placed on the image plane

  - compute coordinate (u, v) of each pixel (i, j) on the image plane of size $(r - l) \times (t - b)$

$$u = l + (i + 0.5)\frac{r - l}{n_x}$$
$$v = b + (j + 0.5)\frac{t - b}{n_y}$$

  - set ray's origin to be $\mathbf{e} + u\mathbf{u} + v\mathbf{v}$ and direction to be $-\mathbf{w}$

(b) **(perspective views)** project along lines that pass through a single point, the viewpoint, rather than along parallel lines. $\mathbf{e}$ the viewpoint is placed some distance d in front of $\mathbf{e}$, call this distance the **image plane distance / focal length**

  - compute coordinate (u, v) of each pixel (i, j) on the image plane using previous formula
  - set ray's origin to be $\mathbf{e}$ and direction to be $-d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$

from eye $\mathbf{e}$ to a point $\mathbf{s}$ on the image plane.

7. **(ray-sphere intersection)** Solve a quadratic formula satisfying the 2 condition, that the intersecting point is both on the ray and on the sphere.

8. **(ray-triangle intersection)** Want to find the first intersection between the ray $\mathbf{e} + t\mathbf{d}$ and a surface that occurs at a $t \in [t_0, t_1]$. Given parametric surfaces , we can solve for

$$\mathbf{e} + t\mathbf{d} = \mathbf{f}(u, v)$$

for 3 unknowns, u, u, v. If the surface is a plane in barycentric coordinate, then solve for

$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

for some t, β, γ. The intersection is inside the triangle if and only if $\beta > 0$,$\gamma > 0$, and $\beta + \gamma < 1$. There is no solution if either the triangle is degenerate or the ray is parallel to the plane containing the triangle. This equation can be solved analytically with Cramer's rule

9. **(ray-polygon intersection)** Given a planar polygon with vertices $\{\mathbf{p}_1, \cdots, \mathbf{p}_m\}$ and surface normal $\mathbf{n}$, we can compute intersection point between ray and plane containing polygon $\mathbf{p}$ with

$$(\mathbf{p} - \mathbf{p}_1) \cdot n = 0 \qquad t = \frac{(\mathbf{p}_1 - \mathbf{e}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

then we check if $\mathbf{p}$ is inside the polygon by sending 2D ray out from $\mathbf{p}$ and count the number of intersections between the ray and boundary of the polygon: if the number of intersections is odd, then the point is inside the polygon.

10. **(ray-scene-intersection)** To intersect a ray with a group of objects, i.e. the scene, simply intersect ray with the objects in the group and return the intersection with the smallest t value.

11. **(shading model)** is designed to capture the process of light reflection, whereby surfaces are illuminated by light sources and reflect part of the light to the camera

  - (light direction) $\mathbf{l}$ is unit vector pointing toward light source
  - (view direction) $\mathbf{v}$ is the unit vector pointing toward camera
  - (surface normal) $\mathbf{n}$ is unit surface normal at point of reflection
  - (properties of the surface), i.e. color, shininess

12. **(Lambertian Shading)** For each color channel, compute pixel color L with

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l})$$

where k_d is the diffuse coefficient or the surface color, and I is intensity of light source, and $n \cdot l = \cos(\theta)$ where θ is angle between surface normal and light source. Lambertian shading is **view independent**, i.e. color of the surface does not depend on the viewing direction $\mathbf{v}$, leading to matte, chalky appearance. Higher k_d, more contrasty

13. **(Blinn-Phong Shading)** To account for shininess, or **specular reflection**, a **specular component** is added to the **diffuse component** to account for highlights. Blinn-Phong accounts for specular reflection by generating brightest reflection when $\mathbf{v}$ and $\mathbf{l}$ are symemtrically positioned across the surface normal, i.e. mirror reflection; the reflection decreases smoothly as the vectors move away from the mirror configuration. Idea is to compare bisector of $\mathbf{v}$ and $\mathbf{l}$ to the surface normal $\mathbf{n}$

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^p \quad \mathbf{h} = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

where k_s is the specular coefficient of the surface and p is Phong exponent where larger value indicate shinier/glossier surface

14. **(Ambient Shading)** To avoid rendering completely black pixels for surfaces that receive no illumination, add a constant illumination to surfaces, with no dependence on surface geometry

$$L = k_a I_a + k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

where k_a is surface specific ambient coefficient and I_a is the ambient light intensity.

15. **(multiple point lights)** the effect by more than one light source is simply the sum of the effects of the light sources individually by superposition. Hence

$$L = k_a I_a + \sum_{i=1}^{N} \left(k_d I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p \right)$$

where $I_i, \mathbf{l}_i, \mathbf{h}_i$ are intensity, direction, and half vector of the i-th light source

16. **(shadows)** idea is surface is illuminated if nothing blocks the view of the light, i.e. the shadow ray $\mathbf{p} + t\mathbf{l}$ does not intersect any object in the scene.

17. **(ideal specular reflection)** or mirror reflection of a surface. the viewer sees the reflection of other objects instead of the highlights. The mirrored object's color is the color of ray at the surface in the reflection direction

5 Linear Algebra

1. **(eigenvalues and diagonalization)** If a matrix has eigenvectors, then we can find them by solving a quadratic equation

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = 0$$

2. **(eigenvalue decomposition)** If $\mathbf{A}$ is symmetric, i.e. $\mathbf{A} = \mathbf{A}^T$, then we can decompose

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T$$

where $\mathbf{Q}$ is an orthogonal matrix whose column are eigenvectors of $\mathbf{A}$ and $\mathbf{D}$ is a diagonal matrix whose diagonals are eigenvalues of $\mathbf{A}$

3. **(single value decomposition)** Any A can be decomposed to

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

where $\mathbf{U}, \mathbf{V}$ are orthogonal matrices, whose columns are left,right singular vectors of $\mathbf{A}$ and $\mathbf{S}$ is a diagonal matrix containing singular values of $\mathbf{A}$. Let $\mathbf{M} = \mathbf{A}\mathbf{A}^T$,

$$\mathbf{M} = \mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{S}\mathbf{V}^T)(\mathbf{U}\mathbf{S}\mathbf{V}^T)^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T$$

since $\mathbf{M}$ is symmetric so svd reduces to eigenvalue decomposition. Therefore singlular values for $\mathbf{A}$ are square roots of eigenvalues of $\mathbf{M}$ and singular vector for $\mathbf{A}$ are eigenvectors for $\mathbf{M}$

6 Transformation Matrices

1. **(2D linear transformations)** scaling, shearing, reflection, rotation, etc. talks bout how symmetric eigenvalue decomposition can decompose a matrix into rotation, scaling, and rotation back to previous direction. and how singular value decomposition also has a geometric meaning, except the two rotation are not the same

2. **(3D linear transforations)**

   - **(scaling)**

   $$scale(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

   - **(rotation)** about one of x,y,z axis.

   $$rotate\_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

   - **(arbitrary rotations)**
   - **(transforming normal vectors)** surface normals are perpendicular to the tangent plane of a surface. Let $\mathbf{t}$ be any tangent vector and $\mathbf{n}$ be surface normal. $\mathbf{t}_M = \mathbf{Mt}$ will still be tangent to the transformed surface. However $\mathbf{Mn}$ may not be perpendicular to the transformed surface. However we can find some matrix $\mathbf{N}$ that transform surface normal correctly

   $$\mathbf{n}^T\mathbf{t} = 0 \quad \mathbf{t}_M = \mathbf{Mt} \quad \mathbf{n}_N = \mathbf{Nn} \quad \mathbf{n}_N^T\mathbf{t}_M = 0$$

   $$\mathbf{n}^T\mathbf{t} = 0 \quad \rightarrow \quad (\mathbf{n}^T\mathbf{M}^{-1})(\mathbf{Mt}) = (\mathbf{n}^T\mathbf{M}^{-1})\mathbf{t}_M = 0 \quad \rightarrow \quad \mathbf{n}_N = \left(\mathbf{n}^T\mathbf{M}^{-1}\right)^T = (\mathbf{M}^{-1})^T\mathbf{n}$$

   therefore $N = (M^{-1})^T$

3. **(Translation and Affine Transformation)** affine transformation is linear transformation followed by a translation. We an represent a 2D affine transformation on position vector with a 3×3 matrix by sing homomeneous coordinate

   $$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

   Matrix multiplication can be used to compose affine transformations. For vector that represent directions or offsets, vectors should not change when we translate the object, we set the third coordinate to zero to accomodate this

   $$\begin{pmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$

   Another way uses shearing in a higher dimension to represent translation. More detail in book

4. **(windowing transformation)** Create transformation matrices that takes point in rectangle $[x_l, x_h] \times [y_l, y_h]$ to rectangle $[x'_l \times x'_h, y'_l \times y'_h]$, which can be accomplished by translation to origin, scale, then translate back

   $$window = translate(x'_l, y'_l) \cdot scale(\frac{x'_h - x'_l}{x_h - x_l}, \frac{y'_h - y'_l}{y_h - y_l}) \cdot translate(-x_l, -y_l)$$

   For 3D windowing transformation from $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$ to $[x'_l, x'_h] \times [y'_l, y'_h] \times [z'_l, z'_h]$ is given by

   $$\begin{pmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. **(Inverses of Transformation Matrices)** certain types of transformation matrices are easy to invert. scaling are diagonal matrices. rotation are orthogonal matricies so inverse is simply the transpose. For arbitrary matrix, we can apply SVD and decompose into rotation, scaling, and another rotation, take inverses of each and then compose them back.

6. **(coordinate transformations)** Let $\mathbf{o}, (\mathbf{x}, \mathbf{y})$ be canonical coordinate system in 2D and let $\mathbf{e}, (\mathbf{u}, \mathbf{v})$ be another coordinate frame. We can specify a **frame-to-canonical matrix** for the (u, v) frame, taking points expressed in (u, v) frame and converts them to same points expressed in the canonical frame.

$$\mathbf{p}_{xy} = \begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{pmatrix} \mathbf{p}_{uv}$$

The **canonical-to-frame matrix** is the inverse

$$\mathbf{p}_{uv} = \begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{pmatrix}^{-1} \mathbf{p}_{xy}$$

Similarly for 3D...

7 Viewing

1. **(viewing transformations)** Moving object between their 3D positions (world space) and their positions in a 2D view of the 3D world (image space). Important to object-order rendering. Idea is we can find matrices that can project any point on a given pixel's viewing ray back to that pixel's position in the image space

2. **(viewing transformations)** map 3D location w.r.t. (x, y, z) coordinate in the canonical coordinate system to coordinates in the image w.r.t. pixels

   (a) (camera/eye transformation) is a rigid body transformation that places the camera at the origin in a convenient orientation, depending on position and orientation, or pose, of the camera

   (b) (projection transformation) projects points from camera space so that all visible points fall in the unit cube $x \in [-1, 1]^3$, only depending on the type of projection desired

   (c) (viewport/windowing transformation) maps unit image rectangle to desired rectangle in pixel coordinate, depending on the size and position of the output image

   The camera transformation converts points in <u>world space</u> to <u>camera space</u>. The projection transformation converts points in camera space to <u>canonial view volume</u>. The viewport transformation maps canonical view volume to <u>screen space</u> Assume we wish to view a scene with an orthographic camera looking along $-z$ direction with $+y$ up

3. **(viewport transformation)** maps axis-aligned rectangle to another, i.e. $[-1, 1]^2 \to [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$ where n_x, n_y are number of pixels for the image. Recall windowing transformation, we have a matrix that ignores z-coordinate of points in the canonical view volume, since a point's distance along the projection direction doesn't affect where the point projects in the image.

$$M_{vp} = \begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   the matrix is called the viewport matrix M_{vp}, where it leaves z-coordinates of points unmodifieds

4. **(orthographic projection transformation)** Instead of the canonical view volume, we want to accomodate arbitrary n-orthotope for orthographic projection. Given coordinates of the side of an axis-aligned volume, called the orthographic view volume, of size $[l, r] \times [b, t] \times [f, n]$, we want a map to the canonical view volume. This is again a windowing transformation

$$M_{orth} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

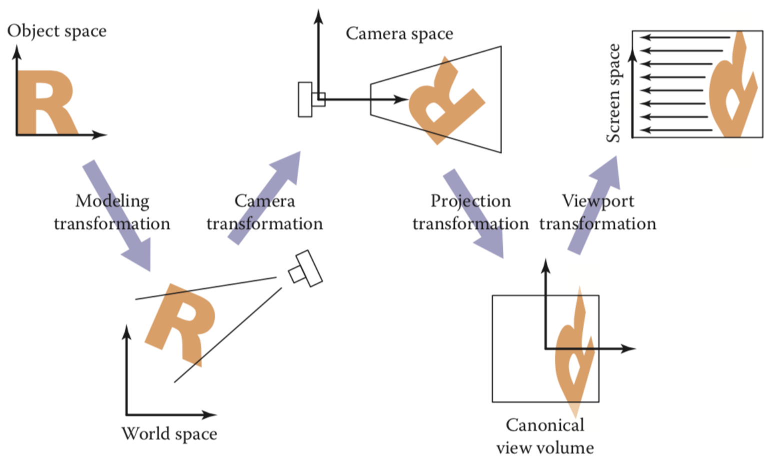

Figure 7.2. The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

Figure 1: viewing_transformations

5. **(the camera transformation)** convention is to use $\mathbf{e}$ or eye position, $\mathbf{g}$ for gaze direction and $\mathbf{t}$ for view-up vector. We can construct a righ-handed basis with

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|} \quad \mathbf{u} = \frac{\mathbf{t}\times\mathbf{w}}{\|\mathbf{t}\times\mathbf{w}\|} \quad \mathbf{v} = \mathbf{w}\times\mathbf{u}$$

idea is $\mathbf{w}$ acts similarly to $-\mathbf{z}$. Now we have a camera coordinate given by $\mathbf{e}, (\mathbf{u}, \mathbf{v}, \mathbf{w})$. We want point's coordinate be with respect to the camera coordinate system, however the points coordinate is in world space. The camera transformation is simply the world-to-frame transformation

$$M_{cam} = \begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

To sum up,

$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ z_{canonical} \\ 1 \end{pmatrix} (M_{vp} M_{orth} M_{cam}) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$