mathematics for 3d game programming and computer graphics

Mathematics for 3D Game Programming and Computer Graphics is an essential foundation for anyone looking to delve into the realms of game design and rendering techniques. Understanding mathematical concepts not only enhances the aesthetic quality of 3D graphics but also optimizes performance in real-time applications like video games. This article will explore the various mathematical principles that underpin 3D game programming, covering essential topics such as vector mathematics, transformations, projection, lighting, and collision detection.

1. Vector Mathematics

Vectors are fundamental in 3D graphics and game programming, representing points, directions, and velocities.

1.1 Definition of Vectors

- A vector is an entity that has both magnitude and direction.
- In a 3D space, a vector can be represented as $\mathbf{v} = (x, y, z)$, where $x, y,$ and z are the vector's components along the three axes.

1.2 Vector Operations

Several operations can be performed on vectors, which are crucial for game programming:

1. Addition: The sum of two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:
$$
\mathbf{c} = \mathbf{a} + \mathbf{b} = (a_x + b_x, a_y + b_y, a_z + b_z)
$$

2. Subtraction: The difference between two vectors is:
$$
\mathbf{c} = \mathbf{a} - \mathbf{b} = (a_x - b_x, a_y - b_y, a_z - b_z)
$$

3. Dot Product: The dot product measures the angle between two vectors:
$$
\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z
$$
- Useful for determining angles and projections.

4. Cross Product: The cross product results in a vector perpendicular to both input vectors:
$$
```
```

$$
\mathbf{c} = \mathbf{a} \times \mathbf{b} = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)
$$
- Important for calculating normals in lighting calculations.

2. Transformations

Transformations are vital in positioning, rotating, and scaling objects in 3D space.

2.1 Types of Transformations

Transformations can be represented using matrices, which allows for efficient computations:

1. Translation: Moving an object from one position to another. The translation matrix is:
$$
T = \begin{pmatrix}
1 & 0 & 0 & tx \\
0 & 1 & 0 & ty \\
0 & 0 & 1 & tz \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

2. Rotation: Rotating an object around an axis. Rotation matrices for the x, y, and z axes are:

- X-axis:
$$
R_x = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) & 0 \\
0 & \sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

- Y-axis:
$$
R_y = \begin{pmatrix}
\cos(\theta) & 0 & \sin(\theta) & 0 \\
0 & 1 & 0 & 0 \\
-\sin(\theta) & 0 & \cos(\theta) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

- Z-axis:
$$
R_z = \begin{pmatrix}

$$
\cos(\theta) & -\sin(\theta) & 0 & 0 \\
\sin(\theta) & \cos(\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

3. Scaling: Changing the size of an object. The scaling matrix is:
$$
S = \begin{pmatrix}
sx & 0 & 0 & 0 \\
0 & sy & 0 & 0 \\
0 & 0 & sz & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

2.2 Applying Transformations

To apply transformations, we multiply the object's position vector by the transformation matrix. For a position vector $\mathbf{p} = (x, y, z, 1)$, the transformed position $\mathbf{p'}$ is computed as:
$$
\mathbf{p'} = M \cdot \mathbf{p}
$$
where M is the combination of translation, rotation, and scaling matrices.

3. Projection

Projection is the process of converting 3D coordinates into 2D coordinates suitable for display on a screen.

3.1 Types of Projection

1. Orthographic Projection: This projection preserves parallel lines and does not account for perspective. The orthographic projection matrix is:
$$
P_{ortho} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

2. Perspective Projection: This projection simulates depth by making objects appear smaller as they are farther from the camera. The perspective projection matrix is:
$$
P_{persp} = \begin{pmatrix}
\frac{1}{\tan(\frac{fov}{2})} & 0 & 0 & 0 \\
0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\
0 & 0 & \frac{far + near}{near - far} & \frac{2 \cdot far \cdot near}{near - far} \\
0 & 0 & -1 & 0
\end{pmatrix}
$$

3.2 View Transformation

To simulate a camera in a 3D scene, view transformations are used. The view matrix transforms world coordinates into camera coordinates, allowing the game to render the scene from the player's perspective. This transformation often involves translating and rotating the scene so that the camera is at the origin, looking down the negative z-axis.

4. Lighting and Shading

Lighting is crucial for creating realistic images in 3D graphics. It involves several mathematical calculations to determine how light interacts with surfaces.

4.1 Lighting Models

1. Ambient Lighting: Provides a base level of light to simulate indirect lighting.
2. Diffuse Lighting: Depends on the angle between the light source and the surface normal, calculated using the dot product.
3. Specular Lighting: Accounts for shiny spots on surfaces, which depend on the viewer's position relative to the light source and surface.

The Phong reflection model combines these components, providing a formula for the final color of a pixel:
$$
I = I_a + I_d + I_s
$$
where I_a is the ambient light, I_d is the diffuse light, and I_s is the specular light.

4.2 Normal Vectors

Normal vectors are essential for lighting calculations. They are perpendicular to the surface and are used to determine how light interacts with that surface. For flat surfaces, the normal can be calculated using the cross product of two edge vectors.

5. Collision Detection

Collision detection is a critical aspect of game programming that determines whether two objects intersect.

5.1 Bounding Volumes

Using bounding volumes simplifies collision detection significantly. Common types include:

- Axis-Aligned Bounding Boxes (AABB): Rectangles in 3D space aligned with the coordinate axes.
- Bounding Spheres: Defined by a center point and a radius.

5.2 Intersection Tests

1. AABB-AABB: Check if the bounding boxes overlap in all three axes.
2. Sphere-Sphere: Compare the distance between centers to the sum of the radii.
3. Ray Casting: A method of determining if a ray intersects with an object, useful for line-of-sight calculations and shooting mechanics.