Game Programming

Robin Bing-Yu Chen
National Taiwan University
Game Mathematics

- Vectors
- Matrices
- Transformations
- Homogeneous Coordinates
- 3D Viewing
- Triangle Mathematics
- Intersection Issues
- Fixed-point Real Numbers
- Quaternions
- Parametric Curves
Vectors

- A vector is an entity that possesses \textit{magnitude} and \textit{direction}.
- A ray (directed line segment), that possesses \textit{position, magnitude}, and \textit{direction}.

\[
\mathbf{v} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)
\]

\[
(x_1, y_1, z_1)
\]

\[
(x_2, y_2, z_2)
\]
Vectors

- an n-tuple of real numbers (scalars)
- two operations: addition & multiplication
- Commutative Laws
  - $a + b = b + a$
  - $a \cdot b = b \cdot a$
- Identities
  - $a + 0 = a$
  - $a \cdot 1 = a$
- Associative Laws
  - $(a + b) + c = a + (b + c)$
  - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Distributive Laws
  - $a \cdot (b + c) = a \cdot b + a \cdot c$
  - $(a + b) \cdot c = a \cdot c + b \cdot c$
- Inverse
  - $a + b = 0 \rightarrow b = -a$
Addition of Vectors

- parallelogram rule

\[ \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 7 \end{pmatrix} \]
The Vector Dot (Inner) Product

\[ u = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad v = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \]

\[ \Rightarrow u \cdot v = x_1 y_1 + \ldots + x_n y_n \]

- length = \( \sqrt{u \cdot u} = \|u\| \)
Properties of the Dot Product

- **symmetric**
  - \( v \cdot w = w \cdot v \)

- **nondegenerate**
  - \( v \cdot v = 0 \) only when \( v = 0 \)

- **bilinear**
  - \( v \cdot (u + \alpha w) = v \cdot u + \alpha (v \cdot w) \)

- **unit vector (normalizing)**
  - \( v' = v / \|v\| \)

- **angle between the vectors**
  - \( \cos^{-1} \left( \frac{v \cdot w}{\|v\| \|w\|} \right) \)
Projection

\[ |u| = |w| \cos \theta \]

\[ = |w| \left( \frac{v' \cdot w}{|v'| |w|} \right) \]

\[ = v' \cdot w \]
Cross Product of Vectors

**Definition**

\[ x = v \times w \]

\[ = (v_2w_3 - v_3w_2)i + (v_3w_1 - v_1w_3)i + (v_1w_2 - v_2w_1)k \]

where \( i = (1, 0, 0) \), \( j = (0,1,0) \), \( k = (0, 0, 1) \) are standard unit vectors.

**Application**

A normal vector to a polygon is calculated from 3 (non-collinear) vertices of the polygon.
Vector in Unity

- **Vector2**
  - Representation of 2D vectors and points (e.g., texture coordinates in a Mesh or texture offsets in Material).
  - In the majority of other cases a Vector3 is used.

- **Vector2Int**
  - using integers.
Vector in Unity

- Vector3
  - Representation of 3D vectors and points.
  - this structure is used throughout Unity to pass 3D positions and directions around.

- Vector3Int
  - using integers.

- Vector4
  - Representation of four-dimensional vectors. (e.g., mesh tangents, parameters for shaders)
Vector3 in Unity

- **Static Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>Shorthand for writing Vector3(0, 0, -1).</td>
</tr>
<tr>
<td>down</td>
<td>Shorthand for writing Vector3(0, -1, 0).</td>
</tr>
<tr>
<td>forward</td>
<td>Shorthand for writing Vector3(0, 0, 1).</td>
</tr>
<tr>
<td>left</td>
<td>Shorthand for writing Vector3(-1, 0, 0).</td>
</tr>
<tr>
<td>one</td>
<td>Shorthand for writing Vector3(1, 1, 1).</td>
</tr>
<tr>
<td>right</td>
<td>Shorthand for writing Vector3(1, 0, 0).</td>
</tr>
<tr>
<td>up</td>
<td>Shorthand for writing Vector3(0, 1, 0).</td>
</tr>
<tr>
<td>zero</td>
<td>Shorthand for writing Vector3(0, 0, 0).</td>
</tr>
</tbody>
</table>
# Vector3 in Unity

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnitude</td>
<td>Returns the length of this vector (Read Only).</td>
</tr>
<tr>
<td>normalized</td>
<td>Returns this vector with a magnitude of 1 (Read Only).</td>
</tr>
<tr>
<td>sqrMagnitude</td>
<td>Returns the squared length of this vector (Read Only).</td>
</tr>
<tr>
<td>this[int]</td>
<td>Access the x, y, z components using [0], [1], [2] respectively.</td>
</tr>
<tr>
<td>x</td>
<td>X component of the vector.</td>
</tr>
<tr>
<td>y</td>
<td>Y component of the vector.</td>
</tr>
<tr>
<td>z</td>
<td>Z component of the vector.</td>
</tr>
</tbody>
</table>
## Vector3 in Unity

### Static Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Angle</strong></td>
<td>Returns the angle in degrees between from and to.</td>
</tr>
<tr>
<td><strong>ClampMagnitude</strong></td>
<td>Returns a copy of vector with its magnitude clamped to maxLength.</td>
</tr>
<tr>
<td><strong>Cross</strong></td>
<td>Cross Product of two vectors.</td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>Returns the distance between a and b.</td>
</tr>
<tr>
<td><strong>Dot</strong></td>
<td>Dot Product of two vectors.</td>
</tr>
<tr>
<td><strong>Lerp</strong></td>
<td>Linearly interpolates between two vectors.</td>
</tr>
<tr>
<td><strong>MoveTowards</strong></td>
<td>Moves a point current in a straight line towards a target point.</td>
</tr>
<tr>
<td><strong>Normalize</strong></td>
<td>Makes this vector have a magnitude of 1.</td>
</tr>
<tr>
<td><strong>OrthoNormalize</strong></td>
<td>Makes vectors normalized and orthogonal to each other.</td>
</tr>
</tbody>
</table>
Vector3 in Unity

- **Static Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Projects a vector onto another vector.</td>
</tr>
<tr>
<td>ProjectOnPlane</td>
<td>Projects a vector onto a plane defined by a normal orthogonal to the plane.</td>
</tr>
<tr>
<td>Reflect</td>
<td>Reflects a vector off the plane defined by a normal.</td>
</tr>
<tr>
<td>RotateTowards</td>
<td>Rotates a vector current towards target.</td>
</tr>
<tr>
<td>Slerp</td>
<td>Spherically interpolates between two vectors.</td>
</tr>
</tbody>
</table>
Matrix Basics

- **Definition**
  \[
  A = (a_{ij}) = \begin{bmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{bmatrix}
  \]

- **Transpose**
  \[
  C = A^T, \quad c_{ij} = a_{ji} \implies C = \begin{bmatrix} a_{11} & \ldots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \ldots & a_{nm} \end{bmatrix}
  \]

- **Addition**
  \[
  C = A + B, \quad c_{ij} = a_{ij} + b_{ij}
  \]
Matrix Basics

- Scalar-matrix multiplication
  \[ C = \alpha A \quad c_{ij} = \alpha a_{ij} \]

- Matrix-matrix multiplication
  \[ C = AB \quad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \]

- Matrix multiplication are not commutative
  \[ AB \neq BA \]
Matrix in Unity

- **Matrix4x4**
  - You rarely use matrices in scripts; most often using Vector3s, Quaternions and functionality of Transform class is more straightforward.
  - Setting up nonstandard camera projection.
  - In Unity, Matrix4x4 is used by several Transform, Camera, Material and GL functions.
    - Transform.localToWorldMatrix
    - Transform.worldToLocalMatrix
    - Camera.projectionMatrix
    - Camera.worldToCameraMatrix
    - ...

General Transformations

A transformation maps points to other points and/or vectors to other vectors

\[ v = T(u) \]

\[ Q = T(P) \]
Pipeline Implementation

$T$ (from application program)

transformation

$T(u)$

$T(v)$

rasterizer

frame buffer

$T(v)$

vertices

$T(u)$

pixels
We can represent a point, \( p = (x, y) \) in the plane
- as a column vector \[
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]
- as a row vector \[
\begin{bmatrix}
x & y
\end{bmatrix}
\]
2D Transformations

- 2D Translation
- 2D Scaling
- 2D Reflection
- 2D Shearing
- 2D Rotation
Translation

- Move (translate, displace) a point to a new location

- Displacement determined by a vector $d$
  - Three degrees of freedom
  - $P' = P + d$
2D Translation

\[ P' = P + T \]
\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = 
\begin{bmatrix}
  x \\
  y
\end{bmatrix} + 
\begin{bmatrix}
  d_x \\
  d_y
\end{bmatrix}
\]

(4,5) \rightarrow (7,1)

(7,5) \rightarrow (10,1)
2D Scaling

\[ P' = S \cdot P \]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  s_x & 0 \\
  0 & s_y
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]
2D Reflection

\[ P' = R E_x \cdot P \]

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} = \begin{bmatrix}
    -1 & 0 \\
    0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]
2D Shearing

\[
\begin{align*}
P' &= SH_x \cdot P \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\end{align*}
\]
2D Rotation

\[
P' = R \cdot P
\]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{bmatrix} \cdot \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

(4,5) (7,5) → (2.1,4.9) (4.9,7.8)
2D Rotation

- Consider rotation about the origin by $\theta$ degrees
  - radius stays the same, angle increases by $\theta$

\[
x' = r \cos (\phi + \theta)
\]
\[
y' = r \sin (\phi + \theta)
\]

\[
x = r \cos \phi
\]
\[
y = r \sin \phi
\]
Limitations of a 2X2 matrix

- Scaling
- Rotation
- Reflection
- Shearing

- What do we miss?
Homogeneous Coordinates

Why & What is homogeneous coordinates?

- If points are expressed in homogeneous coordinates, all three transformations can be treated as multiplications.

\[(x, y) \rightarrow (x, y, W)\]

- Usually 1 can not be 0
Homogeneous Coordinates

$\mathbf{P} = \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$

$W = 1$ plane
Homogeneous Coordinates for 2D Translation

\[
P' = P + T
\]
\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
x \\
y
\end{bmatrix} + \begin{bmatrix}
d_x \\
d_y
\end{bmatrix}
\]

\[
P' = T(d_x, d_y) \cdot P
\]
\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & d_x \\
0 & 1 & d_y \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]

\[
P'' = T(d_{x2}, d_{y2}) \cdot P'
\]
Homogeneous Coordinates for 2D Translation

\[ P'' = T(d_{x2}, d_{y2}) \bullet (T(d_{x1}, d_{y1}) \bullet P) \]

\[ = (T(d_{x2}, d_{y2}) \bullet T(d_{x1}, d_{y1})) \bullet P \]

\[ T(d_{x2}, d_{y2}) \bullet T(d_{x1}, d_{y1}) = \begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} \]

\[ = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \]
Homogeneous Coordinates for 2D Scaling

\[
P' = S \cdot P
\]

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

\[
P' = S(s_{x_2}, s_{y_2}) \cdot S(s_{x_1}, s_{y_1}) =
\begin{bmatrix}
    s_{x_2} & 0 & 0 \\
    0 & s_{y_2} & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    s_{x_1} & 0 & 0 \\
    0 & s_{y_1} & 0 \\
    0 & 0 & 1
\end{bmatrix}
\]

\[
= 
\begin{bmatrix}
    s_{x_1} \cdot s_{x_2} & 0 & 0 \\
    0 & s_{y_1} \cdot s_{y_2} & 0 \\
    0 & 0 & 1
\end{bmatrix}
\]
Homogeneous Coordinates for 2D Rotation

\[ P' = R \cdot P \]

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} = \begin{bmatrix}
    \cos \theta & -\sin \theta \\
    \sin \theta & \cos \theta
\end{bmatrix} \cdot \begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

\[ P' = R(\theta) \cdot P \]

\[
\begin{bmatrix}
    x' \\
    y' \\
    1
\end{bmatrix} = \begin{bmatrix}
    \cos \theta & -\sin \theta & 0 \\
    \sin \theta & \cos \theta & 0 \\
    0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]
Properties of Transformations

- rigid-body transformations
  - rotation & translation
  - preserving angles and lengths
- affine transformations
  - rotation & translation & scaling
  - preserving parallelism of lines
Composition of 2D Transformations

Original

After translation of $P_1$ to origin

$P_1 = (x_1, y_1)$

$T(-x_1, -y_1)$

$R(\theta)$

After translation to original $P_1$

$T(x_1, y_1)$
Composition of 2D Transformations

\[
T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & x_1(1-\cos \theta) + y_1 \sin \theta \\ \sin \theta & \cos \theta & y_1(1-\cos \theta) - x_1 \sin \theta \\ 0 & 0 & 1 \end{bmatrix}
\]
Right-handed Coordinate System
3D Translation & 3D Scaling

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

\[ S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
3D Reflection & 3D Shearing

\[
RE_x = \begin{bmatrix}
-1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
RE_y = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
SH_{xy}(sh_x, sh_y) = \begin{bmatrix}
1 & 0 & sh_x & 0 \\
0 & 1 & sh_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
3D Rotations

\[ R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Rotation About a Fixed Point other than the Origin

- Move fixed point to origin
- Rotate
- Move fixed point back
- $M = T(P_f) \cdot R(\theta) \cdot T(-P_f)$
A rotation by $\theta$ about an arbitrary axis can be decomposed into the concatenation of rotations about the $x$, $y$, and $z$ axes.

- $\theta_x, \theta_y, \theta_z$ are called the Euler angles.

$$R(\theta) = R_z(\theta_z) \cdot R_y(\theta_y) \cdot R_x(\theta_x)$$

Note that rotations do not commute.
- We can use rotations in another order but with different angles.
Transform in Unity

- **Transform**
  - Every object in a Scene has a Transform.
  - Position, rotation and scale of an object.
  - Hierarchically structure

```csharp
using UnityEngine;

class Example : MonoBehaviour {
    void Start() {
        foreach (Transform child in transform) {
            child.position += Vector3.up * 10.0f;
        }
    }
}
```
# Transform in Unity

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>The blue axis of the transform in world space.</td>
</tr>
<tr>
<td>right</td>
<td>The red axis of the transform in world space.</td>
</tr>
<tr>
<td>up</td>
<td>The green axis of the transform in world space.</td>
</tr>
<tr>
<td>position</td>
<td>The position of the transform in world space.</td>
</tr>
<tr>
<td>rotation</td>
<td>The rotation of the transform in world space stored as a Quaternion.</td>
</tr>
<tr>
<td>lossyScale</td>
<td>The global scale of the object (Read Only).</td>
</tr>
<tr>
<td>localPosition</td>
<td>Position of the transform relative to the parent transform.</td>
</tr>
<tr>
<td>localRotation</td>
<td>The rotation of the transform relative to the transform rotation of the parent.</td>
</tr>
<tr>
<td>localScale</td>
<td>The scale of the transform relative to the parent.</td>
</tr>
</tbody>
</table>
## Transform in Unity

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>root</strong></td>
<td>Returns the topmost transform in the hierarchy.</td>
</tr>
<tr>
<td><strong>parent</strong></td>
<td>The parent of the transform.</td>
</tr>
<tr>
<td><strong>childCount</strong></td>
<td>The number of children the parent Transform has.</td>
</tr>
<tr>
<td><strong>localToWorldMatrix</strong></td>
<td>Matrix that transforms a point from local space into world space (Read Only).</td>
</tr>
<tr>
<td><strong>worldToLocalMatrix</strong></td>
<td>Matrix that transforms a point from world space into local space (Read Only).</td>
</tr>
</tbody>
</table>
# Transform in Unity

## Public Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Translate</strong></td>
<td>Moves the transform in the direction and distance of translation.</td>
</tr>
<tr>
<td><strong>Rotate</strong></td>
<td>Applies a rotation of eulerAngles.z degrees around the z axis, eulerAngles.x degrees around the x axis, and eulerAngles.y degrees around the y axis (in that order).</td>
</tr>
<tr>
<td><strong>RotateAround</strong></td>
<td>Rotates the transform about axis passing through point in world coordinates by angle degrees.</td>
</tr>
<tr>
<td><strong>LookAt</strong></td>
<td>Rotates the transform so the forward vector points at /target/’s current position.</td>
</tr>
</tbody>
</table>

# Transform in Unity

## Public Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>InverseTransformVector</code></td>
<td>Transforms a vector from world space to local space. The opposite of <code>Transform.TransformVector</code>.</td>
</tr>
<tr>
<td><code>SetPositionAndRotation</code></td>
<td>Sets the world space position and rotation of the <code>Transform</code> component.</td>
</tr>
<tr>
<td><code>TransformDirection</code></td>
<td>Transforms direction from local space to world space.</td>
</tr>
<tr>
<td><code>TransformPoint</code></td>
<td>Transforms position from local space to world space.</td>
</tr>
<tr>
<td><code>TransformVector</code></td>
<td>Transforms vector from local space to world space.</td>
</tr>
</tbody>
</table>
# Transform in Unity

## Public Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DetachChildren</td>
<td>Unparents all children.</td>
</tr>
<tr>
<td>Find</td>
<td>Finds a child by n and returns it.</td>
</tr>
<tr>
<td>GetChild</td>
<td>Returns a transform child by index.</td>
</tr>
<tr>
<td>GetSiblingIndex</td>
<td>Gets the sibling index.</td>
</tr>
<tr>
<td>IsChildOf</td>
<td>Is this transform a child of parent?</td>
</tr>
<tr>
<td>SetAsFirstSibling</td>
<td>Move the transform to the start of the local transform list.</td>
</tr>
<tr>
<td>SetAsLastSibling</td>
<td>Move the transform to the end of the local transform list.</td>
</tr>
<tr>
<td>SetParent</td>
<td>Set the parent of the transform.</td>
</tr>
<tr>
<td>SetSiblingIndex</td>
<td>Sets the sibling index.</td>
</tr>
</tbody>
</table>
Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*).
- Drawing simple perspectives by hand uses these vanishing point(s).
Triangular Coordinate System

\[ h = \frac{A_a}{A} h_a + \frac{A_b}{A} h_b + \frac{A_c}{A} h_c \]

where \( A = A_a + A_b + A_c \)

if \((A_a < 0 \parallel A_b < 0 \parallel A_c < 0)\) than the point is outside the triangle
Triangular Coordinate System - Application

- Terrain following
  - Interpolating the height of arbitrary point within the triangle

- Hit test
  - Intersection of a ray from camera to a screen position with a triangle

- Ray cast
  - Intersection of a ray with a triangle

- Collision detection
  - Intersection
Intersection

- Ray cast
- Containment test
Ray Cast – The Ray

- Cast a ray to calculate the intersection of the ray with models
- Use parametric equation for a ray

\[
\begin{align*}
  x &= x_0 + (x_1 - x_0)t, \\
  y &= y_0 + (y_1 - y_0)t, \\
  z &= z_0 + (z_1 - z_0)t, \quad t \geq 0
\end{align*}
\]

- When \( t = 0 \), the ray is on the start point \((x_0, y_0, z_0)\)
- Only the \( t \geq 0 \) is the answer candidate
- The smallest positive \( t \) is the answer
Ray Cast – The Plane

- Each triangle in the 3D models has its plane equation.
- Use $Ax + By + Cz + D = 0$ as the plane equation.
- $(A, B, C)$ is the plane normal vector.
- $|D|$ is the distance of the plane to origin.
- Substitute the ray equation into the plane.
- Solve the $t$ to find the intersect point.
- Check the intersect point within the triangle or not by using “Triangle Area Test”.
2D Containment Test

- If the no. of intersection is odd, the point is inside, otherwise, is outside.

\[ \text{Intersection} = 0, \text{ outside} \]
\[ \text{Intersection} = 1, \text{ inside} \]
\[ \text{Intersection} = 2, \text{ outside} \]

- If the no. of intersection is odd, the point is inside, otherwise, is outside.
3D Containment Test

- if the no. of intersection is odd, the point is inside, otherwise, is outside
Ray Cast in Unity

to the closest object

RaycastHit hitInfo = new RaycastHit();
Vector3 dir = new Vector3(-1, 0, 0);
if (Physics.Raycast(this.transform.position, dir, out hitInfo)) {
  if (hitInfo.collider.gameObject.name == "CubeA") {
    print("shoot");
  }
}
Ray Cast in Unity

☐ to all objects

```csharp
Vector3 dir = new Vector3(-1, 0, 0);
RaycastHit[] hitInfos =
Physics.RaycastAll(this.transform.position, dir);
foreach (RaycastHit hitInfo in hitInfos) {
    print(hitInfo.collider.gameObject.name);
}
```
Tagging Objects for Ray Cast in Unity

- Tag旗標是用來識別和分類物件的方法之一
- 在Ray Cast時能用於識別物件
- 設定Tag的方法在Inspector視窗中的Tag中點擊Add Tag...之後會出現TagManager，拉下Tags之後就能加入新的Tag
Ray Cast for a Tag in Unity

Vector3 dir = new Vector3(-1, 0, 0);
RaycastHit[] hitInfos =
Physics.RaycastAll(this.transform.position, dir);
foreach (RaycastHit hitInfo in hitInfos) {
    if (hitInfo.collider.gameObject.tag != "Boom")
        print(hitInfo.collider.gameObject.name);
}
Ray Cast from Camera’s view in Unity

RaycastHit hit = new RaycastHit();
Vector3 pos = Input.mousePosition;
Ray mouseray = Camera.main.ScreenPointToRay(pos);
if (Input.GetMouseButton(0)) {
    if (Physics.Raycast(mouseray, out hit)) {
        ......
    }
}
Fixed Point Arithmetic

- Fixed point arithmetic: \( n \) bits (signed) integer
  - Example: \( n = 16 \) gives range \(-32768 \leq \tilde{a} \leq 32767\)
  - We can use fixed scale to get the decimals.

\[
\begin{align*}
\text{8 integer bits} &\quad 11111111110  \\
\text{8 fractional bits} &\quad \tilde{a} = 1600 \rightarrow a = 6.25
\end{align*}
\]
Fixed Point Arithmetic

- Multiplication requires rescaling

\[ e = a \cdot c = \tilde{a} \cdot 2^{-8} \cdot \tilde{c} \cdot 2^{-8} = \tilde{e} \cdot 2^{-8} \]

\[ \Rightarrow \tilde{e} = (\tilde{a} \cdot \tilde{c}) \cdot 2^{-8} \]

- Addition just like normal

\[ e = a + c = \tilde{a} \cdot 2^{-8} + \tilde{c} \cdot 2^{-8} = (\tilde{a} + \tilde{c}) \cdot 2^{-8} \]

\[ \Rightarrow \tilde{e} = \tilde{a} + \tilde{c} \]
Fixed Point Arithmetic - Application

- Compression for floating-point real numbers
  - 4 bytes reduced to 2 bytes
  - Lost some accuracy but affordable

- Network data transfer

- Software 3D rendering (without hardware-assistant)
Euler Angles

- An Euler angle is a rotation about a single axis.
- A rotation is described as a sequence of rotations about three mutually orthogonal coordinates axes fixed in space
  - X-roll, Y-roll, Z-roll
- There are 6 possible ways to define a rotation.
  - 3!
Interpolating Euler Angles

- **Natural orientation representation:**
  - 3 angles for 3 degrees of freedom

- **Unnatural interpolation:**
  - A rotation of $90^\circ$ first around Z and then around Y = $120^\circ$ around $(1, 1, 1)$.
  - But $30^\circ$ around Z then Y differs from $40^\circ$ around $(1, 1, 1)$.
Incremental Rotation

- Consider the two approaches
  - For a sequence of rotation matrices $R_0, R_1, \ldots, R_n$, find the Euler angles for each and use $R_i = R_{iz} R_{iy} R_{ix}$
  - Not very efficient
  - Use the final positions to determine the axis and angle of rotation, then increment only the angle
- Quaternions can be more efficient than either
Solution: Quaternion Interpolation

- Interpolate orientation on the unit sphere
- By analogy: 1-, 2-, 3-DOF rotations as constrained points on 1-, 2-, 3-spheres
1D-Sphere and Complex Plane

- Interpolate orientation in 2D
- 1 angle
  - but messy because modulo $2\pi$
- Use interpolation in (complex) 2D plane
- Orientation = complex argument of the number
Quaternions

- Quaternions are unit vectors on 3-sphere (in 4D)
- Right-hand rotation of $\theta$ radians about $\mathbf{v}$ is $q = [\cos(\theta/2), \sin(\theta/2) \cdot \mathbf{v}]$
  - often noted $[\mathbf{w}, \mathbf{v}]$
- Requires one real and three imaginary components $i, j, k$
  - $q = q_0 + q_1 i + q_2 j + q_3 k = [\mathbf{w}, \mathbf{v}]$; $\mathbf{w} = q_0$, $\mathbf{v} = (q_1, q_2, q_3)$
  - where $i^2 = j^2 = k^2 = ijk = -1$
  - $\mathbf{w}$ is called **scalar** and $\mathbf{v}$ is called **vector**
Basic Operations Using Quaternions

- **Addition**  \( q + q' = [w + w', v + v'] \)
- **Multiplication**  \( q \cdot q' = [w \cdot w' - v \cdot v', v \times v' + w \cdot v' + w' \cdot v] \)
- **Conjugate**  \( q^* = [w, -v] \)
- **Length**  \( |q| = (w^2 + |v|^2)^{1/2} \)
- **Norm**  \( N(q) = |q|^2 = w^2 + |v|^2 \)
- **Inverse**  \( q^{-1} = q^*/|q|^2 = q^*/N(q) \)
- **Unit Quaternion**
  - \( q \) is a unit quaternion if \( |q| = 1 \) and then \( q^{-1} = q^* \)
- **Identity**
  - \([1, (0, 0, 0)]\) (when involving multiplication)
  - \([0, (0, 0, 0)]\) (when involving addition)
SLERP-Spherical Linear intERPolation

- Interpolate between two quaternion rotations along the shortest arc.

\[ \text{SLERP}(p, q, t) = \frac{p \cdot \sin((1-t)\theta) + q \cdot \sin(t\theta)}{\sin(\theta)} \]

- Where \( \cos(\theta) = w_p \cdot w_q + v_p \cdot v_q \)

- If two orientations are too close, use linear interpolation to avoid any divisions by zero.
Quaternion in Unity

- They are compact, don't suffer from gimbal lock and can easily be interpolated. Unity internally uses Quaternions to represent all rotations.
- never access or modify individual Quaternion components \((x,y,z,w)\)
- 99% of the time are:
  - Quaternion.LookRotation
  - Quaternion.Angle
  - Quaternion.Euler
  - Quaternion.Slerp
  - Quaternion.FromToRotation
  - Quaternion.identity
Parametric Polynomial Curves

- We will use parametric curves where the functions are all polynomials in the parameter.

\[ x(u) = \sum_{k=0}^{n} a_k u^k \]

\[ y(u) = \sum_{k=0}^{n} b_k u^k \]

- Advantages:
  - easy (and efficient) to compute
  - infinitely differentiable
Parametric Cubic Curves

- Fix $n = 3$
- The cubic polynomials that define a curve segment $Q(t) = [x(t)\ y(t)\ z(t)]^T$ are of the form

\[
\begin{align*}
    x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\
    y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\
    z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \quad 0 \leq t \leq 1.
\end{align*}
\]
Parametric Cubic Curves

The curve segment can be rewrite as

\[ Q(t) = [x(t) \quad y(t) \quad z(t)]^T = C \bullet T \]

where \[ T = [t^3 \quad t^2 \quad t \quad 1]^T \]

\[
C = \begin{bmatrix}
    a_x & b_x & c_x & d_x \\
    a_y & b_y & c_y & d_y \\
    a_z & b_z & c_z & d_z
\end{bmatrix}
\]
Tangent Vector

\[
\frac{d}{dt} Q(t) = Q'(t) = \begin{bmatrix}
\frac{d}{dt} x(t) & \frac{d}{dt} y(t) & \frac{d}{dt} z(t)
\end{bmatrix}^T
\]

\[
= \frac{d}{dt} C \bullet T = C \bullet \begin{bmatrix}
3t^2 & 2t & 1 & 0
\end{bmatrix}^T
\]

\[
= \begin{bmatrix}
3a_x t^2 + 2b_x t + c_x & 3a_y t^2 + 2b_y t + c_y & 3a_z t^2 + 2b_z t + c_z
\end{bmatrix}^T
\]
Three Types of Parametric Cubic Curves

- Hermite Curves
  - defined by two endpoints and two endpoint tangent vectors

- Bézier Curves
  - defined by two endpoints and two control points which control the endpoint’ tangent vectors

- Splines
  - defined by four control points
Parametric Cubic Curves

- $Q(t) = C \cdot T$
- rewrite the coefficient matrix as $C = G \cdot M$
  - where $M$ is a 4x4 basis matrix, $G$ is called the geometry matrix
  - so

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

4 endpoints or tangent vectors
Parametric Cubic Curves

\[ Q(t) = G \cdot M \cdot T = G \cdot B \]

where \( B = M \cdot T \) is called the \textbf{blending functions}
Hermite Curves

- Given the endpoints $P_1$ and $P_4$ and tangent vectors at them $R_1$ and $R_4$.

- What is
  - Hermite basis matrix $M_H$
  - Hermite geometry vector $G_H$
  - Hermite blending functions $B_H$

- by definition

$$G_H = \begin{bmatrix} P_1 & P_4 & R_1 & R_4 \end{bmatrix}$$
Hermite Curves

since

\[\begin{align*}
Q(0) &= P_1 = G_H \cdot M_H \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \\
Q(1) &= P_4 = G_H \cdot M_H \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T \\
Q'(0) &= R_1 = G_H \cdot M_H \cdot \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T \\
Q'(1) &= R_4 = G_H \cdot M_H \cdot \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix}^T
\end{align*}\]

\[G_H = \begin{bmatrix} P_1 & P_4 & R_1 & R_4 \end{bmatrix} = G_H \cdot M_H \cdot \begin{bmatrix} 0 & 1 & 0 & 3 \\
0 & 1 & 0 & 2 \\
0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 \end{bmatrix}\]
Hermite Curves

\[
M_H = \begin{bmatrix}
0 & 1 & 0 & 3 \\
0 & 1 & 0 & 2 \\
0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0
\end{bmatrix}^{-1} = \begin{bmatrix}
2 & -3 & 0 & 1 \\
-2 & 3 & 0 & 0 \\
1 & -2 & 1 & 0 \\
1 & -1 & 0 & 0
\end{bmatrix}
\]

\[
Q(t) = G_H \bullet M_H \bullet T = G_H \bullet B_H
\]

\[
B_H = \begin{bmatrix}
2t^3 - 3t^2 + 1 \\
-2t^3 + 3t^2 \\
t^3 - 2t^2 + t \\
t^3 - t^2
\end{bmatrix}^T
\]
Computing a point

- Given two endpoints $P_1$ and $P_4$ and two tangent vectors at them $R_1$ and $R_4$

So

$$Q(t) = \begin{bmatrix} P_1 & P_4 & R_1 & R_4 \end{bmatrix} \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$
Bézier Curves

- Given the endpoints $P_1$ and $P_4$ and two control points $P_2$ and $P_3$ which determine the endpoints’ tangent vectors, such that
  
  $R_1 = Q'(0) = 3(P_2 - P_1)$

  $R_4 = Q'(1) = 3(P_4 - P_3)$

- What is
  - Bézier basis matrix $M_B$
  - Bézier geometry vector $G_B$
  - Bézier blending functions $B_B$
Bézier Curves

- by definition \( G_B = [P_1 \ P_2 \ P_3 \ P_4] \)
- then \( G_H = [P_1 \ P_4 \ R_1 \ R_4] \)

\[
\begin{bmatrix}
1 & 0 & -3 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & -3 \\
0 & 1 & 0 & 3
\end{bmatrix} = G_B \cdot M_{HB}
\]

- so \( Q(t) = G_H \cdot M_H \cdot T = (G_B \cdot M_{HB}) \cdot M_H \cdot T \)

\[
= G_B \cdot (M_{HB} \cdot M_H) \cdot T = G_B \cdot M_B \cdot T
\]
Bézier Curves

and

\[ M_B = M_{HB} \cdot M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

\[ Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \]

\[ B_B = [(1-t)^3 \quad 3t(1-t)^2 \quad 3t^2(1-t) \quad t^3]^T \]

Bernstein polynomials
Bernstein Polynomials

- The coefficients of the control points are a set of functions called the Bernstein polynomials: 
  \[ Q(t) = \sum_{i=0}^{n} b_i(t) P_i \]

- For degree 3, we have:
  \[ b_0(t) = (1 - t)^3 \]
  \[ b_1(t) = 3t(1 - t)^2 \]
  \[ b_2(t) = 3t^2 (1 - t) \]
  \[ b_3(t) = t^3 \]
Bernstein Polynomials

- Useful properties on the interval $[0,1]$:
  - each is between 0 and 1
  - sum of all four is exact 1
  - a.k.a., a “partition of unity”

- These together imply that the curve lines within the convex hull of its control points.
Convex Hull
Subdividing Bézier Curves

- \( Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2 (1-t) P_3 + t^3 P_4 \)
- How to draw the curve?
- How to convert it to be line-segments?
Subdividing Bézier Curves (de Casteljau’s algorithm)

- \( Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4 \)

- How to draw the curve?
- How to convert it to be line-segments?

\[
Q\left(\frac{1}{2}\right) = \frac{1}{8} P_1 + \frac{3}{8} P_2 + \frac{3}{8} P_3 + \frac{1}{8} P_4 \\
= \frac{1}{2} \left( \frac{1}{2} (P_1 + P_2) + \frac{1}{2} (P_2 + P_3) \right) + \frac{1}{2} \left( \frac{1}{2} (P_3 + P_4) + \frac{1}{2} (P_2 + P_3) \right)
\]
Display Bézier Curves

DisplayBezzer(P1,P2,P3,P4)

begin
  if (FlatEnough(P1,P2,P3,P4))
  Line(P1,P4);
  else
  Subdivide(P[])=>L[],R[]
  DisplayBezzer(L1,L2,L3,L4);
  DisplayBezzer(R1,R2,R3,R4);
end;
Testing for Flatness

- Compare total length of control polygon to length of line connecting endpoints

\[
\frac{|P_1 - P_2| + |P_2 - P_3| + |P_3 - P_4|}{|P_1 - P_4|} < 1 + \varepsilon
\]
What do we want for a curve?

- Local control
- Interpolation
- Continuity
Local Control

- One problem with Bézier curve is that every control points affect every point on the curve (except for endpoints). Moving a single control point affects the whole curve.
- We’d like to have local control, that is, have each control point affect some well-defined neighborhood around that point.
Bézier curves are approximating. The curve does not necessarily pass through all the control points. We’d like to have a curve that is interpolating, that is, that always passes through every control point.
Continuity between Curve Segments
Continuity between Curve Segments

- $G^0$ geometric continuity
  - two curve segments join together

- $G^1$ geometric continuity
  - the directions (but not necessarily the magnitudes) of the two segments’ tangent vectors are equal at a join point
Continuity between Curve Segments

- $C^1$ continuous
  - the tangent vectors of the two cubic curve segments are equal \((\text{both directions and magnitudes})\) at the segments’ join point

- $C^n$ continuous
  - the direction and magnitude of \(\frac{d^n}{dt^n}[Q(t)]\) through the \(n\)th derivative are equal at the join point
Continuity between Curve Segments

\[ y(t) \]

\[ x(t) \]

join point

\[ C_0 \]

\[ C_1 \]

\[ C_2 \]
Continuity between Curve Segments

\[ y(t) \]

\[ \mathcal{T} V_1 = \mathcal{T} V_2 \]

\[ \mathcal{T} V_3 \]

\[ P_1, P_2, Q_1, Q_2, Q_3, P_3 \]
Bézier Curves → Splines

- Bézier curves have C-infinity continuity on their interiors, but we saw that they do not exhibit local control or interpolate their control points.
- It is possible to define points that we want to interpolate, and then solve for the Bézier control points that will do the job.
- But, you will need as many control points as interpolated points -> high order polynomials -> wiggly curves. (And you still won’t have local control.)
Bézier Curves → Splines

- We will splice together a curve from individual Bézier segments. We call these curves **splines**.
- When splicing Bézier together, we need to worry about continuity.
Ensuring $C^0$ continuity

- Suppose we have a cubic Bézier defined by $(V_1, V_2, V_3, V_4)$, and we want to attach another curve $(W_1, W_2, W_3, W_4)$ to it, so that there is $C^0$ continuity at the joint.

$$C^0 : Q_V(1) = Q_W(0)$$

- What constraint(s) does this place on $(W_1, W_2, W_3, W_4)$?

$$Q_V(1) = Q_W(0) \Rightarrow V_4 = W_1$$
Ensuring $C^1$ continuity

- Suppose we have a cubic Bézier defined by $(V_1, V_2, V_3, V_4)$, and we want to attach another curve $(W_1, W_2, W_3, W_4)$ to it, so that there is $C^1$ continuity at the joint.

\[ C^0 : Q_V(1) = Q_W(0) \]

\[ C^1 : Q'_V(1) = Q'_W(0) \]

- What constraint(s) does this place on $(W_1, W_2, W_3, W_4)$?

\[ Q_V(1) = Q_W(0) \Rightarrow V_4 = W_1 \]

\[ Q'_V(1) = Q'_W(0) \Rightarrow V_4 - V_3 = W_2 - W_1 \]
The $C^1$ Bézier Spline

How then could we construct a curve passing through a set of points $P_1...P_n$?

- We can specify the Bézier control points directly, or we can devise a scheme for placing them automatically...
Catmull-Rom Spline

- If we set each derivative to be one half of the vector between the previous and next controls, we get a **Catmull-Rom Spline**.

- This leads to:
  
  \[ V_1 = P_2 \]
  
  \[ V_2 = P_2 + \frac{1}{6} (P_3 - P_1) \]
  
  \[ V_3 = P_3 - \frac{1}{6} (P_4 - P_2) \]
  
  \[ V_4 = P_3 \]
Catmull-Rom Basis Matrix

\[ Q(t) = G_B \cdot M_B \cdot t \]

\[
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[ = G_B \cdot T \]

\[
G_B = \begin{bmatrix}
V_1 \\
V_2 \\
V_3 \\
V_4
\end{bmatrix}
= \begin{bmatrix}
0 & 1 & 0 & 0 \\
-1 & 1 & 1 & 0 \\
0 & 1 & 1 & -1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
P_1 \\
P_2 \\
P_3 \\
P_4
\end{bmatrix}
\]

\[
Q(t) = [P_1 \ P_2 \ P_3 \ P_4] \frac{1}{2}
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
2 & -5 & 4 & -1 \\
-1 & 0 & 1 & 0 \\
0 & 2 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
t^3 \\
t^2 \\
t \\
1
\end{bmatrix}
\]
Ensuring $C^2$ continuity

Suppose we have a cubic Bézier defined by $(V_1, V_2, V_3, V_4)$, and we want to attach another curve $(W_1, W_2, W_3, W_4)$ to it, so that there is $C^2$ continuity at the joint.

\[
Q'_V(1) = Q'_W(0) \Rightarrow V_4 - V_3 = W_2 - W_1
\]

\[
Q''_V(1) = Q''_W(0) \Rightarrow V_2 - 2V_3 + V_4 = W_1 - 2W_2 + W_3
\]

\[\downarrow\]

\[
W_3 = V_2 - 4V_3 + 4V_4
\]
B-Spline

- Instead of specifying the Bézier control points themselves, let’s specify the corners of the A-frames in order to build a $C^2$ continuous spline.
B-Spline

\[ W_3 = V_2 - 4V_3 + 4V_4 \]

\[ = 2(2V_4 - V_3) - (2V_3 - V_2) \]

\[ = 2W_2 - B_2 \]
B-Spline

Instead of specifying the Bézier control points themselves, let’s specify the corners of the A-frames in order to build a $C^2$ continuous spline.
Instead of specifying the Bézier control points themselves, let’s specify the corners of the A-frames in order to build a $C^2$ continuous spline.

\[
V_1 = \frac{1}{2} \left( B_1 + \frac{2}{3} (B_2 - B_1) + B_2 + \frac{1}{3} (B_3 - B_2) \right)
\]

\[
V_2 = B_2 + \frac{1}{3} (B_3 - B_2)
\]
B-Spline

Instead of specifying the Bézier control points themselves, let’s specify the corners of the A-frames in order to build a $C^2$ continuous spline.

These are called B-Splines. The starting set of points are called de Boor points.
Uniform NonRational B-Splines

- cubic B-Spline
  - has \( m + 1 \) control points \( P_0, P_1, \ldots, P_m, m \geq 3 \)
  - has \( m - 2 \) cubic polynomial curve segments \( Q_3, Q_4, \ldots, Q_m \)

- uniform
  - the knots are spaced at equal intervals of the parameter \( t \)

- non-rational
  - not rational cubic polynomial curves
Uniform NonRational B-Splines

- curve segment $Q_i$ is defined by points $P_{i-3}, P_{i-2}, P_{i-1}, P_i$, thus

- **B-Spline geometry matrix**

$$G_{Bs_i} =\begin{bmatrix} P_{i-3} & P_{i-2} & P_{i-1} & P_i \end{bmatrix} , \quad 3 \leq i \leq m$$

- if $T_i =\begin{bmatrix} (t-t_i)^3 & (t-t_i)^2 & (t-t_i) & 1 \end{bmatrix}^T$

- then $Q_i(t) = G_{Bs_i} \cdot M_{Bs} \cdot T_i , \quad t_i \leq t \leq t_{i+1}$
Uniform NonRational B-Splines

so B-Spline basis matrix

\[
M_{Bs} = \frac{1}{6} \begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 0 & 4 \\
-3 & 3 & 3 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

B-Spline blending functions

\[
B_{Bs} = \frac{1}{6} \begin{bmatrix}
(1-t)^3 & 3t^3 - 6t^2 + 4 & -3t^3 + 3t^2 + 3t + 1 & t^3
\end{bmatrix}^T, \quad 0 \leq t \leq 1
\]
NonUniform NonRational B-Splines

- the **knot-value sequence** is a nondecreasing sequence
- allow **multiple knot** and the number of identical parameter is the **multiplicity**
  - Ex. (0,0,0,0,1,1,2,3,4,4,5,5,5,5,5)

- so

\[ Q_i(t) = P_{i-3} \cdot B_{i-3,4}(t) + P_{i-2} \cdot B_{i-2,4}(t) + P_{i-1} \cdot B_{i-1,4}(t) + P_i \cdot B_{i,4}(t) \]
NonUniform NonRational B-Splines

where $B_{i,j}(t)$ is $j$th-order blending function for weighting control point $P_i$

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leq t \leq t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,2}(t) = \frac{t-t_i}{t_{i+1}-t_i} B_{i,1}(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}} B_{i+1,1}(t)$$

$$B_{i,3}(t) = \frac{t-t_i}{t_{i+2}-t_i} B_{i,2}(t) + \frac{t_{i+3}-t}{t_{i+3}-t_{i+1}} B_{i+1,2}(t)$$

$$B_{i,4}(t) = \frac{t-t_i}{t_{i+3}-t_i} B_{i,3}(t) + \frac{t_{i+4}-t}{t_{i+4}-t_{i+1}} B_{i+1,3}(t)$$
Knot Multiplicity & Continuity

- since $Q(t_i)$ is within the convex hull of $P_{i-3}$, $P_{i-2}$, and $P_{i-1}$
- if $t_i = t_{i+1}$, $Q(t_i)$ is within the convex hull of $P_{i-3}$, $P_{i-2}$, and $P_{i-1}$ and the convex hull of $P_{i-2}$, $P_{i-1}$, and $P_i$, so it will lie on $P_{i-2}P_{i-1}$
- if $t_i = t_{i+1} = t_{i+2}$, $Q(t_i)$ will lie on $P_{i-1}$
- if $t_i = t_{i+1} = t_{i+2} = t_{i+3}$, $Q(t_i)$ will lie on both $P_{i-1}$ and $P_i$, and the curve becomes broken
Knot Multiplicity & Continuity

- multiplicity 1: $C^2$ continuity
- multiplicity 2: $C^1$ continuity
- multiplicity 3: $C^0$ continuity
- multiplicity 4: no continuity
NURBS: NonUniform Rational B-Splines

- rational
  - \( x(t) \), \( y(t) \), and \( z(t) \) are defined as the ratio of two cubic polynomials

- rational cubic polynomial curve segments are ratios of polynomials

\[

dx(t) = \frac{X(t)}{W(t)} \quad dy(t) = \frac{Y(t)}{W(t)} \quad dz(t) = \frac{Z(t)}{W(t)}
\]

- can be Bézier, Hermite, or B-Splines
Parametric Curves in Unity

- No script API supported in standard assets
- AnimationCurve
Parametric Curves in Unity

Assets store

B-Spline Path

Timeline Controllable Path Curve Animation Tool
- Animate GameObject Position along B-Spline Curve in Unity Timeline.
- Edit B-Spline Curve Graphically in SceneView.
- Curve Animation Preview in EditMode.
- Velocity Curve Control of Timeline Clip