We already knew

• Game history
• game design information
• Game engine
What’s Next

• Mathematical concepts (lecture 6-10)
• Collision detection and resolution (lecture 11-16)
• Game AI (lecture 17 - )
Mathematical Concepts
3D modelling, model manipulation and rendering require Maths and Physics

• Typical tasks:
  – How to position objects?
  – How to move and rotate objects
  – How do objects interact?
2D Space

• We will start with a 2D space (simpler) and look at issues involved in
  – Modelling
  – Rendering
  – Transforming the model / view
2D Geometry

• Representation with two axes, usually X (horizontal) and Y (vertical)

• **Origin** of the graph and of the 2D space is where the axes cross (X = Y = 0)

• Points are identified by their coordinates
Viewports

• A **viewport** (or **window**) is a rectangle of pixels representing a view into **world space**

• A viewport has its own coordinate system, which may not match that of the geometry.
  – The axes will usually be X horizontal & Y vertical
    • But don’t have to be – rotated viewports
  – The *scale* of the axes may be different
  – The *direction* of the Y axis may differ.
    • E.g. the geometry may be stored with Y up, but the viewport has Y down.
  – The origin (usually in the corners or centre of the viewport) may not match the geometry origin.
Example of changing coordinate system from world space to viewport space:

\[ P = (20, 15) \] in world space. Where is \( P' \) in viewport space?
Rendering

- **Rendering** is the process of converting geometry into screen pixels
- To render a point:
  - Convert vertex coordinates into viewport space
  - Set the colour of the pixel at those coordinates
  - The colour might be stored with the geometry, or we can use a fixed colour (e.g. black)
Rendering Lines and Shapes

- Need to determine which part of the line is visible, where it meets the viewport edge and how to crop it.

  - In “Ye good old days” this was rather difficult
  - With support from rendering libraries easy
Points and Vectors

• Point: a **location** in space
• Vector: a **direction** in space
What’s the Difference?

• The only difference is “meaning”

• But think about
  – “move a picture to the right”
  – “move a picture up”
  – “move a picture in the direction ...”
    • Vectors specify the direction
Moving an Object

• **Translation** of an object
  – Moving without rotating or reflecting
  – *Apply* a vector to all points of an object
  – Vector specifies **direction** and **magnitude** of translation

![Diagram](attachment:diagram.png)
Vectors

A vector is a *directed line segment*

• The length of the segment is called the *length or magnitude* of vector.

• The direction of the segment is called the *direction* of vector.

• Notations: vectors are usually denoted in bold type, e.g., $\mathbf{a}$, $\mathbf{u}$, $\mathbf{F}$, or underlined, $\underline{a}$, $\underline{u}$, $\underline{F}$.

Same direction, red is twice as long
Translation Recipe

- In order to translate (move) an object in the direction given by a vector \( \mathbf{V} \), move all points.

\[
\mathbf{V} = (x_v, y_v) \\
\mathbf{P} = (x_p, y_p) \\
\mathbf{P}' = (x_p + x_v, y_p + y_v)
\]
Multiplying a Vector by a Number

• Multiplying a vector by a positive **scalar** (positive number) does not change the direction but changes the magnitude.

• Multiplying by a negative number reverses the direction and changes the magnitude.
In Coordinates

• \( \mathbf{v} = (x, y) \) a vector, \( \lambda \) a number

\[ \lambda \cdot \mathbf{v} = (\lambda x, \lambda y) \]

Example:

\[ 2 \cdot (2, 5) = (4, 10) \]
\[ 0.7 \cdot (2, 5) = (1.4, 3.5) \]
\[ -2 \cdot (2, 5) = (-4, -10) \]
From A to B

• Which vector should be applied to move a point from \((x_A, y_A)\) to \((x_B, y_B)\)?

\[(x_B - x_A, y_B - y_A)\]
Sum of Two Vectors

- Two vectors $\mathbf{V}$ and $\mathbf{W}$ are added by placing the beginning of $\mathbf{W}$ at the end of $\mathbf{V}$. 
In Coordinates

Let

\cdot \mathbf{V} = (x_v, y_v)

\cdot \mathbf{W} = (x_w, y_w)

Then

\mathbf{V} + \mathbf{W} = (x_v + x_w, y_v + y_w)
Vector Difference

- $\mathbf{V} - \mathbf{W} = \mathbf{V} + (-1) \cdot \mathbf{W}$
In Coordinates

Let

• \( V = (x_v, y_v) \)
• \( W = (x_w, y_w) \)

Then

\[ V - W = (x_v - x_w, y_v - y_w) \]
Applications

• Apply vector $V$ to an object then apply $W$
  – Apply $V + W$
  – Representing motion as a combination of two

• If $V$ takes you to A, $W$ takes you to B, what takes from A to B?
  – Apply $W - V$
  – Shooting, targeting
From 2D to 3D

- 3D geometry adds an extra axis over 2D geometry
  - This “Z” axis represents “depth”
  - Can choose the “direction” of Z
“Handedness”

- Use thumb (X), index finger (Y) & middle finger (Z) to represent the axes
- Use your left hand and the axes are left-handed, otherwise they are right-handed

Right-Handed System
(Z comes out of the screen)

Left-Handed System
(Z goes in to the screen)
Left- vs Right-Handed

• In mathematics, traditionally, right-handed axes are used

• In computing:
  – DirectX and several graphics applications use left-handed axes
  – OpenGL use right-handed

Neither is better, just a choice
Vectors in 3D

- Still a directed interval
- x, y and z coordinates define a vector

- $\mathbf{v} = (x_v, y_v, z_v)$ a vector, $\lambda$ a number
  $\lambda \cdot \mathbf{v} = (\lambda x_v, \lambda y_v, \lambda z_v)$

- $\mathbf{v} = (x_v, y_v, z_v)$; $\mathbf{w} = (x_w, y_w, z_w)$
  $\mathbf{v} + \mathbf{w} = (x_v+x_w, y_v+y_w, z_v+z_w)$

- $\mathbf{v} = (x_v, y_v, z_v)$; $\mathbf{w} = (x_w, y_w, z_w)$
  $\mathbf{v} - \mathbf{w} = (x_v-x_w, y_v-y_w, z_v-z_w)$
Vectors in jMonkeyEngine

• jME defines two classes for vectors
  – Vector3f
  – Vector2f

• Constructors
  – Vector2f(float x, float y)
  – Vector3f(float x, float y, float z)

• Lots of useful methods (see javadoc)
Translation (setting position) in JME

protected void simpleInitApp() {
    Geometry box = ...;

    Vector3f v = new Vector3f(1, 2, 0);
    box.setLocalTranslation(v);

    rootNode.attachChild(box);
}
Translation And the Scene Graph

• Let’s model a table
Boxes for Tabletop and Legs

Box tableTop = new Box(10, 1, 10);
Box leg1 = new Box(1,5,1);
...
Geometry gTableTop = new
    Geometry("TableTop", tableTop);
gTableTop.setMaterial(mat);
Geometry gLeg1 = new
    Geometry("Leg1", leg1);
gLeg1.setMaterial(mat);
...

Beware of Floats

• If you think that the table top is too thick and change
  
  Box tableTop = new Box(10, 1, 10);

  to

  Box tableTop = new Box(10, 0.3, 10);

  you will see an error:

  The constructor Box(int, double, int) is undefined
Use the “f” word! ☺

Box tableTop = new Box(10, 0.3f, 10);

Many jME methods take “single precision” float numbers as input

No need “double precision”
Position the legs

...  
leg1.setLocalTranslation(7, 0, 7);  
leg2.setLocalTranslation(-7, 0, 7);  
leg3.setLocalTranslation(7, 0,-7);  
leg4.setLocalTranslation(-7, 0,-7);  

Attach all to rootNode
Oops...
A Better Scene Graph

rootNode

table

tableTop

legs

leg1

leg2

leg3

leg4
What are “table” and “legs”

• Internal nodes

Node table = new Node(“Table”);
Node legs = new Node(“Legs”);
...

rootNode

... table

... legs

...
Putting it Together

```javascript
legs.attachChild(gLeg1);
legs.attachChild(gLeg2);
legs.attachChild(gLeg3);
legs.attachChild(gLeg4);

table.attachChild(tableTop);
table.attachChildChild(legs)

rootNode.attachChild(table);
```
But Does It Change the Picture?

No
Transforms Are in All Nodes!

```java
legs.move(0,-5f,0);
```
Summary: Manipulation of Vectors

- **Vector addition**
  - $v + w$

- **Vector difference**
  - $v - w = v + (-w)$

- **Scalar multiplication of vectors**
  - $2v$
  - $(-1)v$
  - $(1/2)V$
  - They remain parallel

- **Vector OP**

- **Vector addition sum**
  - $v + w$

- **Vector OP**
Summary: Vector Arithmetic

\( \mathbf{V}= (x_v, y_v, z_v) \) a vector, \( \lambda \) a number

\[ \lambda \mathbf{V} = (\lambda x_v, \lambda y_v, \lambda z_v) \]

\( \mathbf{V} = (x_v, y_v, z_v) ; \mathbf{W} = (x_w, y_w, z_w) \)

\[ \mathbf{V} + \mathbf{W} = (x_v+x_w, y_v+y_w, z_v+z_w) \]

\[ \mathbf{V} - \mathbf{W} = (x_v-x_w, y_v-y_w, z_v-z_w) \]

*What about a product of \( \mathbf{V} \) and \( \mathbf{W} \)?*  
*And why?*
Summary: Vector Algebra

- \( a + b = b + a \)  
  (commutative law)
- \((a + b) + c = a + (b + c)\)  
  (associative law)
- \( a + 0 = a \)
- \( a + (-a) = 0 \)
- \( \lambda (\mu a) = (\lambda \mu) a \)
- \( (\lambda + \mu)a = \lambda a + \mu a \)
- \( \lambda(a + b) = \lambda a + \lambda b \)
- \( 1a = a \)