COMP220: Graphics & Simulation

10: Simulation & Animation

Learning outcomes

By the end of this week, you should be able to:

- **Recall** the key concepts involved in solving mechanics problems
- **Write** programs which feature realistic physics simulations
- **Describe** how a rigged model is transformed to produce animation

Agenda

Agenda

- Lecture (async):
  - **Recap** the mathematical concepts of Newtonian physics.
  - **Explain** the role of rigging in 3D animation.
  - **Introduce** the scene graph and techniques for enhancing character animation.

Agenda

- Lecture (async):
  - **Recap** the mathematical concepts of Newtonian physics.
  - **Explain** the role of rigging in 3D animation.
  - **Introduce** the scene graph and techniques for enhancing character animation.
- Workshop (sync):
  - **Include** the Bullet Physics library in the OpenGL application.
  - **Explore** physics and animation techniques in code.

FALMOUTH
UNIVERSITY

Newtonian Mechanics

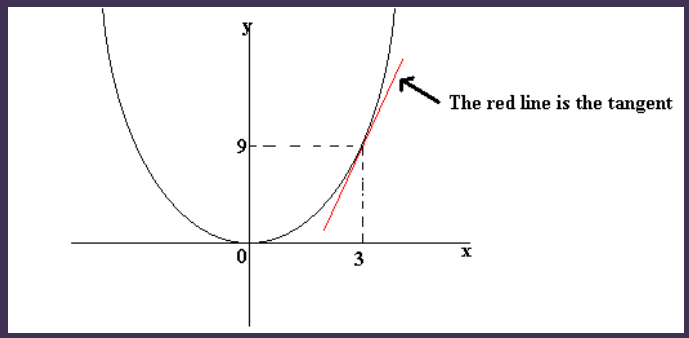
Calculus: differentiation

Calculus: differentiation

- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t

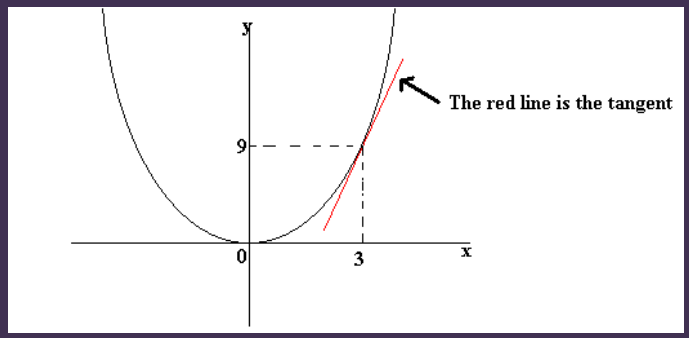
Calculus: differentiation

- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



Calculus: differentiation

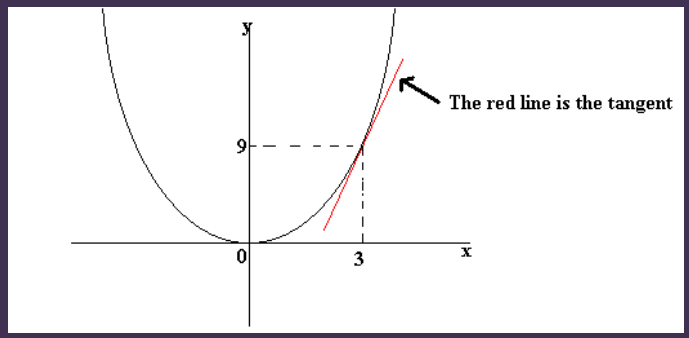
- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



- Equivalent to the **gradient** of a graph, $\frac{\text{change in } y}{\text{change in } x}$

Calculus: differentiation

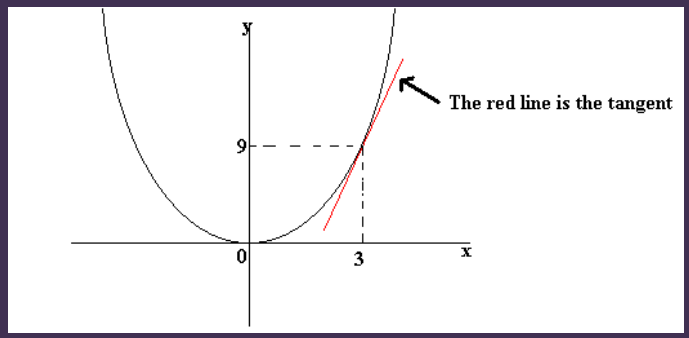
- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



  - Equivalent to the **gradient** of a graph, $\frac{\text{change in } y}{\text{change in } x}$
  - For moving objects:

Calculus: differentiation

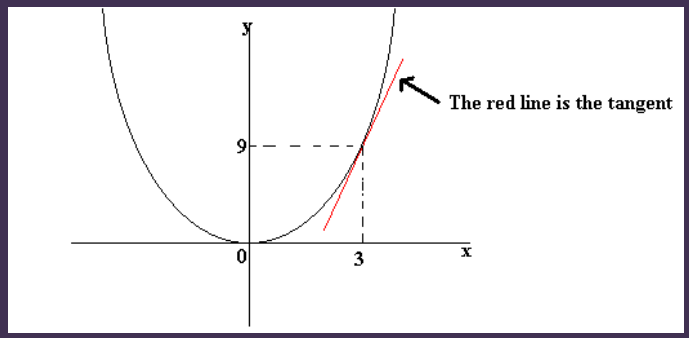
- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



  - Equivalent to the **gradient** of a graph, $\frac{\text{change in } y}{\text{change in } x}$
  - For moving objects:
    - **Velocity** is the derivative of **displacement**

Calculus: differentiation

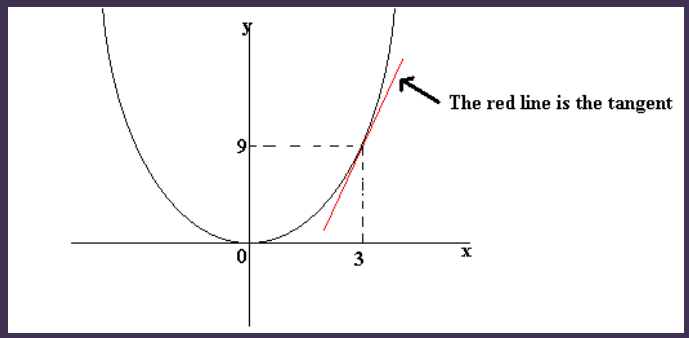
- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



- Equivalent to the **gradient** of a graph, $\frac{\text{change in } y}{\text{change in } x}$
- For moving objects:
  - **Velocity** is the derivative of **displacement**
  - **Acceleration** is the derivative of **velocity**

Calculus: differentiation

- The **derivative** $\frac{dx}{dt}$ of a quantity x with respect to time t is **the rate of change** of x with respect to t
- i.e. how much x changes by if t changes by 1



- Equivalent to the **gradient** of a graph, $\frac{\text{change in } y}{\text{change in } x}$
- For moving objects:
  - **Velocity** is the derivative of **displacement**
  - **Acceleration** is the derivative of **velocity**
  - NB **speed** is the **magnitude** of velocity

Calculus: integration

Calculus: integration

- The opposite of differentiation: x is the **integral** of $\frac{dx}{dt}$

Calculus: integration

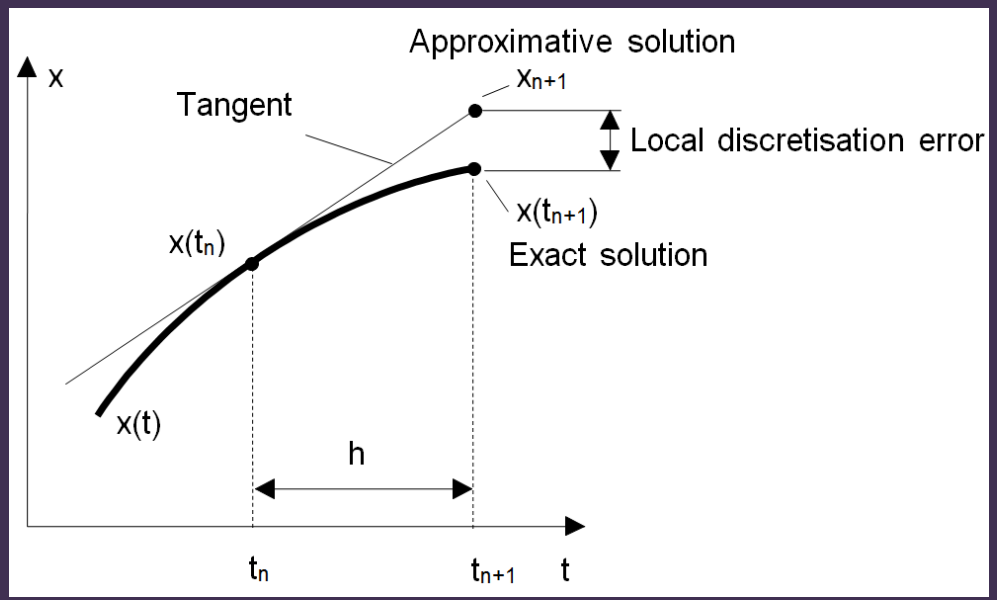
- The opposite of differentiation: x is the **integral** of $\frac{dx}{dt}$
- We are interested in **numerical integration**, i.e. by computer calculation

Calculus: integration

- The opposite of differentiation: x is the **integral** of $\frac{dx}{dt}$
- We are interested in **numerical integration**, i.e. by computer calculation
- **Euler method**: given x and $\frac{dx}{dt}$ at time t, we can **estimate** the value of x at time $t + h$:

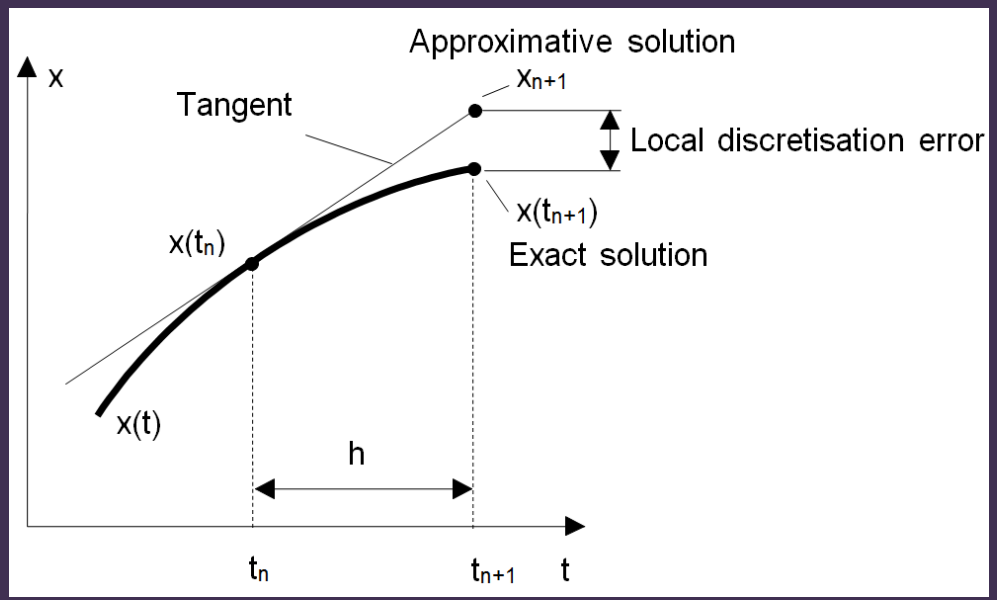
Calculus: integration

- The opposite of differentiation: x is the **integral** of $\frac{dx}{dt}$
- We are interested in **numerical integration**, i.e. by computer calculation
- **Euler method**: given x and $\frac{dx}{dt}$ at time t, we can **estimate** the value of x at time $t + h$:



Calculus: integration

- The opposite of differentiation: x is the **integral** of $\frac{dx}{dt}$
- We are interested in **numerical integration**, i.e. by computer calculation
- **Euler method**: given x and $\frac{dx}{dt}$ at time t, we can **estimate** the value of x at time $t + h$:



$$x(t+h) \approx x(t)+h\times\frac{dx}{dt}(t)$$

Basic simulation loop

Basic simulation loop

- For each object, store its position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$

Basic simulation loop

- For each object, store its position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$
- On each time step Δt:

Basic simulation loop

- For each object, store its position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$
- On each time step Δt:
  - Find the new position using numerical integration, $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{v}\Delta t$

Basic simulation loop

- For each object, store its position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$
- On each time step Δt:
  - Find the new position using numerical integration, $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{v}\Delta t$
  - Calculate the forces acting on the object, and thus the acceleration $\boldsymbol{a}$ from Newton's second law, $\boldsymbol{F} = m\boldsymbol{a}$

Basic simulation loop

- For each object, store its position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$
- On each time step Δt:
  - Find the new position using numerical integration, $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{v}\Delta t$
  - Calculate the forces acting on the object, and thus the acceleration $\boldsymbol{a}$ from Newton's second law, $\boldsymbol{F} = m\boldsymbol{a}$
  - Find the new velocity using numerical integration, $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{a}\Delta t$

Newton's Laws of Motion

Newton's Laws of Motion

An object remains at rest or moves at constant velocity unless acted upon by an external force

Newton's Laws of Motion

An object remains at rest or moves at constant velocity unless acted upon by an external force

$\boldsymbol{F} = m\boldsymbol{a}$: The sum of forces acting upon an object is equal to its mass multiplied by its acceleration

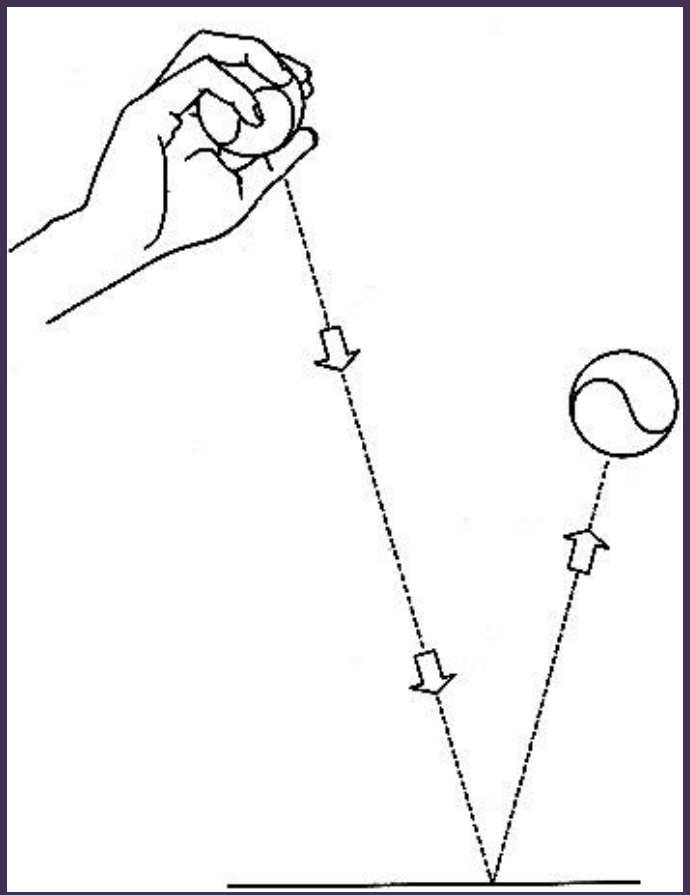
Newton's Laws of Motion

An object remains at rest or moves at constant velocity unless acted upon by an external force

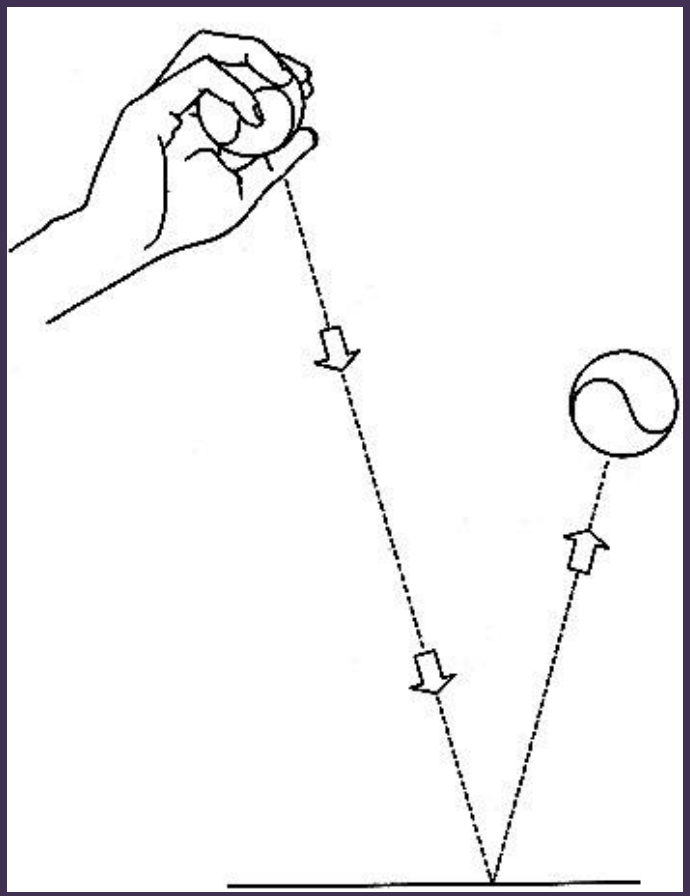
$\boldsymbol{F} = m\boldsymbol{a}$: The sum of forces acting upon an object is equal to its mass multiplied by its acceleration

When one body exerts a force on another, the second body exerts an equal and opposite force on the first

Basic collision response

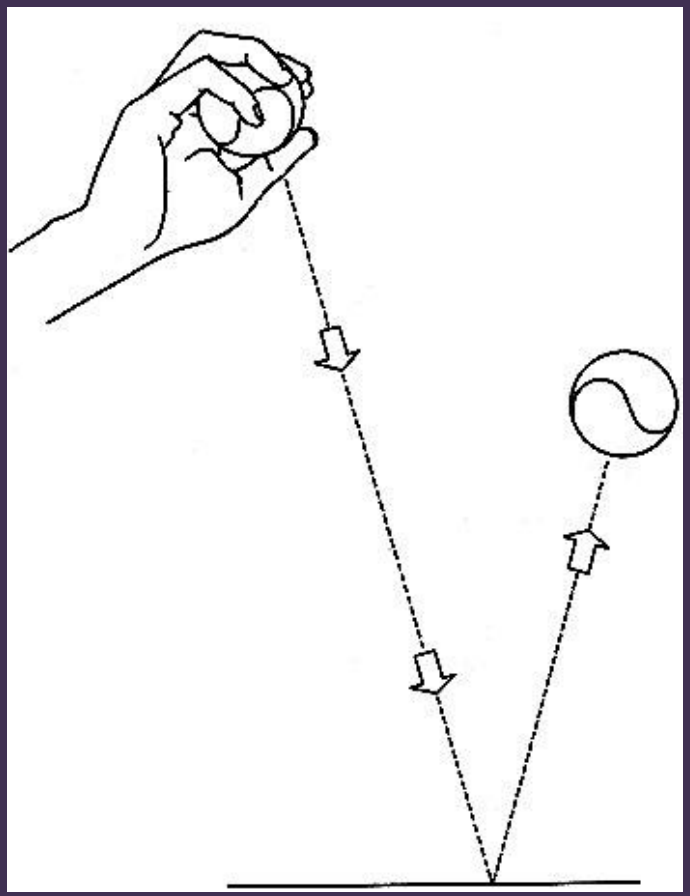


Basic collision response



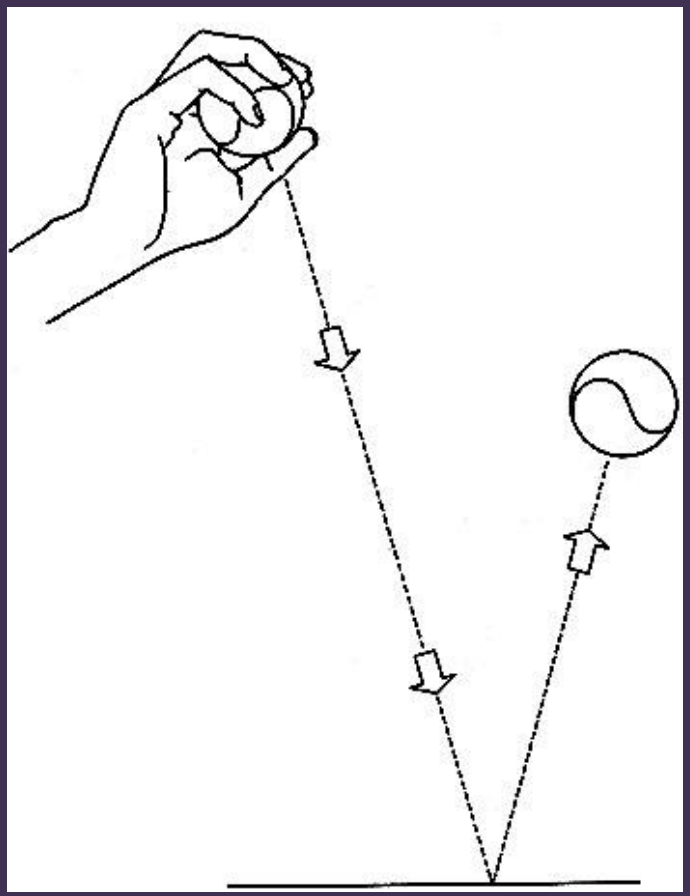
- For an **elastic collision**, the component of velocity parallel to the **surface normal** is **reversed**

Basic collision response



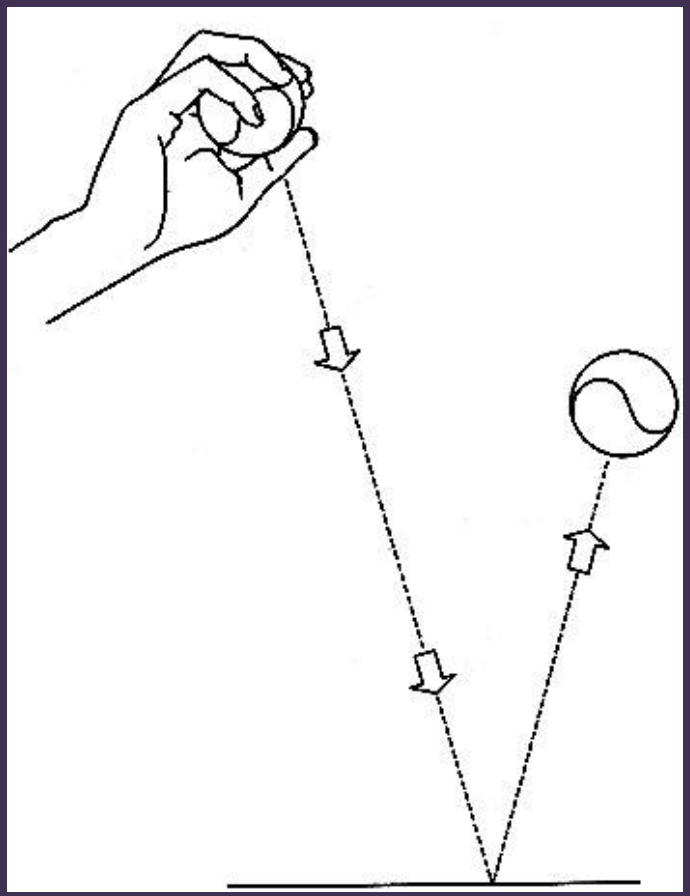
- For an **elastic collision**, the component of velocity parallel to the **surface normal** is **reversed**
- E.g. if the surface is the *xz* plane, flip the *y* component

Basic collision response



- For an **elastic collision**, the component of velocity parallel to the **surface normal** is **reversed**
- E.g. if the surface is the *xz* plane, flip the *y* component
- For an **inelastic collision**, some velocity is lost

Basic collision response



- For an **elastic collision**, the component of velocity parallel to the **surface normal** is **reversed**
- E.g. if the surface is the *xz* plane, flip the *y* component
- For an **inelastic collision**, some velocity is lost
- Flip the *y* component and multiply it by something between 0 and 1

FALMOUTH
UNIVERSITY

Skeletal Animation

Coordinate spaces

Coordinate spaces

Model space

Coordinate spaces

Model space

↓ Model matrix

Coordinate spaces

Model space

↓

World space

Model matrix

Coordinate spaces

Model space

↓ Model matrix

World space

↓ View matrix

Coordinate spaces

Model space

↓ Model matrix

World space

↓ View matrix

Camera space

Coordinate spaces

Model space

↓ Model matrix

World space

↓ View matrix

Camera space

↓ Projection matrix

Coordinate spaces

Model space

↓ Model matrix

World space

↓ View matrix

Camera space

↓ Projection matrix

Screen space

Rule of thumb

Rule of thumb

- When performing calculations, **do not mix** vectors from **different coordinate spaces**

Rule of thumb

- When performing calculations, **do not mix** vectors from **different coordinate spaces**
- E.g. when performing lighting calculations, ensure your fragment position, normal, light direction, eye direction are all in the **same** space

Scene graph

Scene graph

- It is often useful to organise objects into a **hierarchy**

Scene graph

- It is often useful to organise objects into a **hierarchy**
- Each node in the hierarchy has its own model matrix

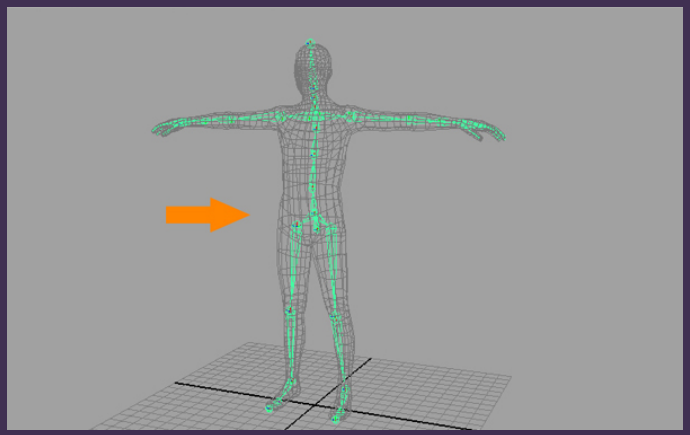
Scene graph

- It is often useful to organise objects into a **hierarchy**
- Each node in the hierarchy has its own model matrix
- Transformations stack: object is affected by its own transformation, and that of its parent, and that of its grandparent, and so on

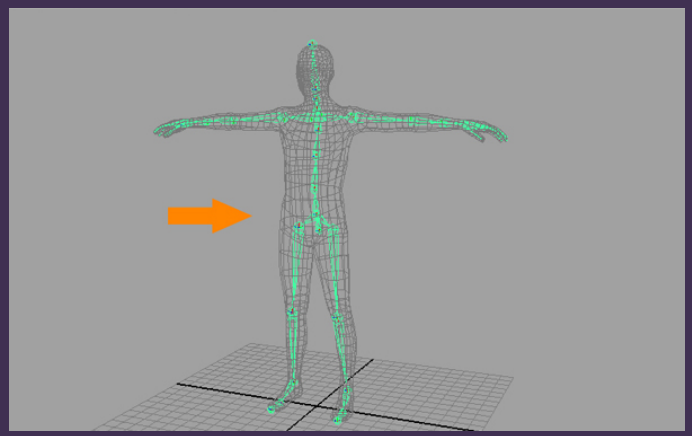
Scene graph

- It is often useful to organise objects into a **hierarchy**
- Each node in the hierarchy has its own model matrix
- Transformations stack: object is affected by its own transformation, and that of its parent, and that of its grandparent, and so on
- The model matrix is the **product** of model matrices for the node and its ancestors

Rigging

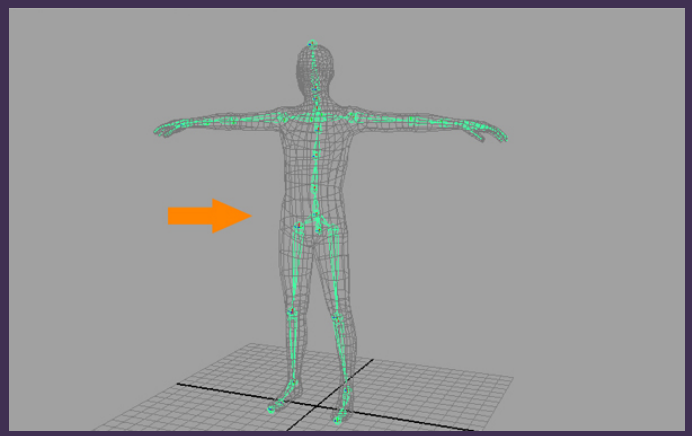


Rigging



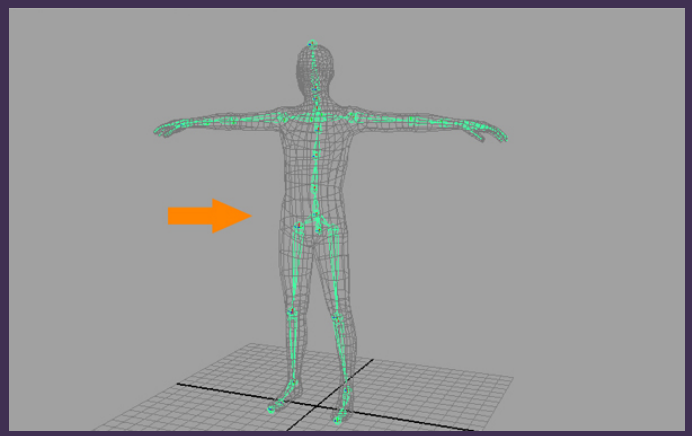
- A **skeleton** is composed of **bones**

Rigging



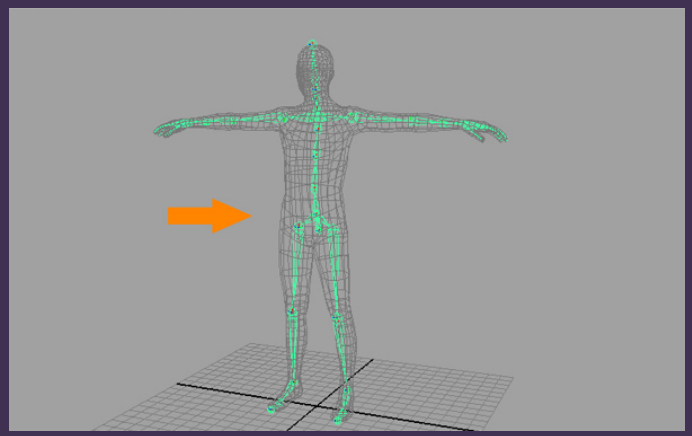
- A **skeleton** is composed of **bones**
- Arranged in a **hierarchy**

Rigging



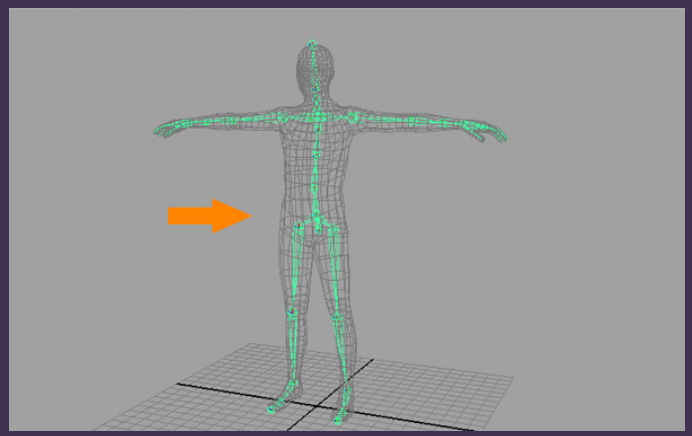
- A **skeleton** is composed of **bones**
- Arranged in a **hierarchy**
- Each bone is essentially just a **transformation**

Rigging



- A **skeleton** is composed of **bones**
- Arranged in a **hierarchy**
- Each bone is essentially just a **transformation**
  - Usually just rotation around a pivot point

Rigging



- A **skeleton** is composed of **bones**
- Arranged in a **hierarchy**
- Each bone is essentially just a **transformation**
  - Usually just rotation around a pivot point
  - 3D modelling software often represents bones as lines from parent bone to child bone

Keyframe animation

Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation

Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation
- ... or just for **keyframes** and interpolate between them

Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation
- ... or just for **keyframes** and interpolate between them
- Keyframes set up by an animator, through motion capture, or a combination of the two

Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation
- ... or just for **keyframes** and interpolate between them
- Keyframes set up by an animator, through motion capture, or a combination of the two
- More advanced: can **blend** animations

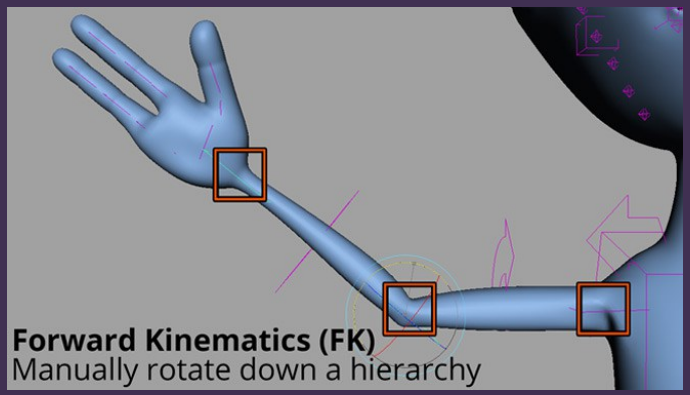
Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation
- ... or just for **keyframes** and interpolate between them
- Keyframes set up by an animator, through motion capture, or a combination of the two
- More advanced: can **blend** animations
  - E.g. blend between walking and running

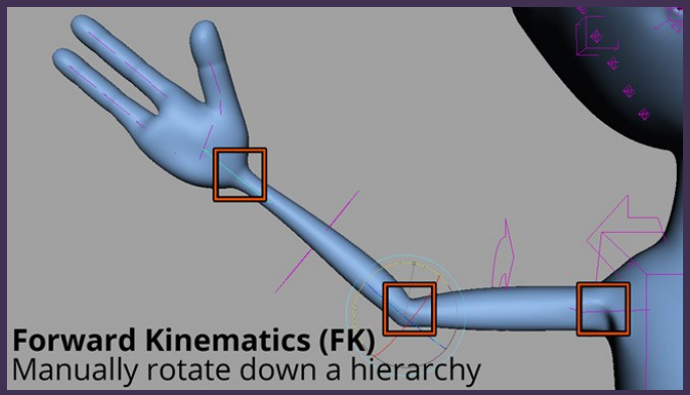
Keyframe animation

- Most basic form of skeletal animation: specify bone transformations for each frame of animation
- ... or just for **keyframes** and interpolate between them
- Keyframes set up by an animator, through motion capture, or a combination of the two
- More advanced: can **blend** animations
  - E.g. blend between walking and running
  - E.g. bottom half plays "walk" animation, top half plays "fire weapon" animation

Forward kinematics (FK)

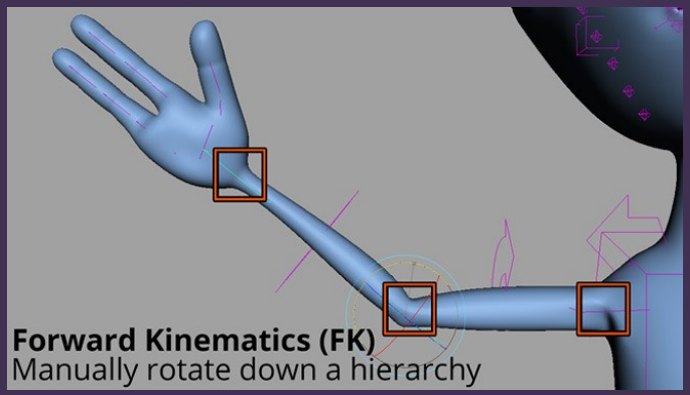


Forward kinematics (FK)



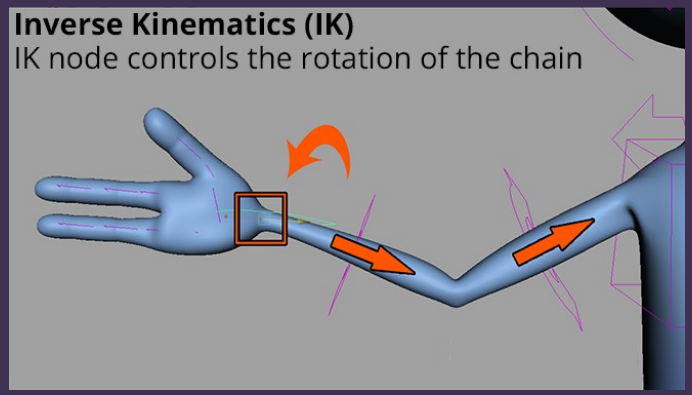
- Bone transformations are set **explicitly**

Forward kinematics (FK)

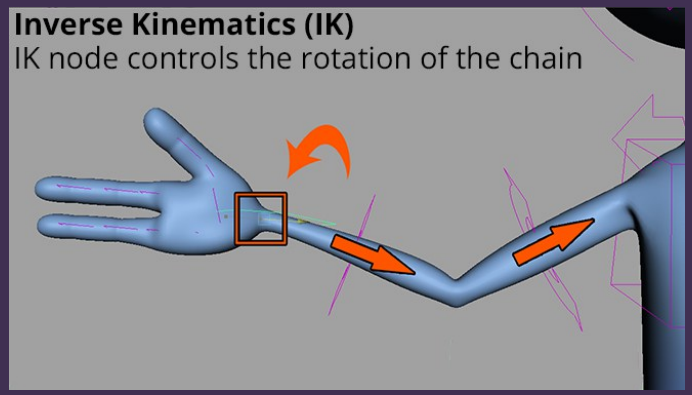


- Bone transformations are set **explicitly**
- Children are affected by parent transformations, e.g. if upper arm rotates, lower arm rotates with it

Inverse kinematics (IK)

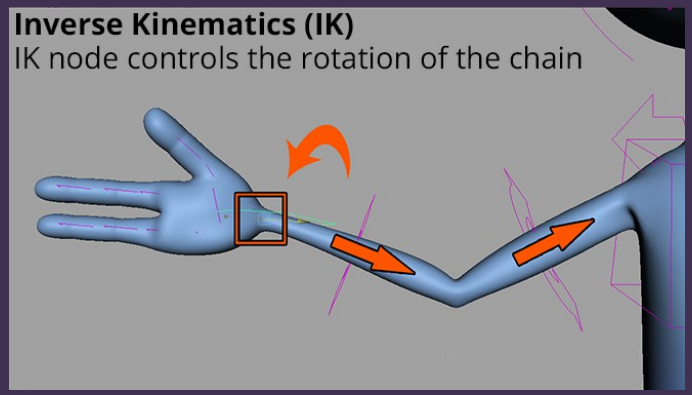


Inverse kinematics (IK)



- Bone transformations are calculated to reach a **target**

Inverse kinematics (IK)



- Bone transformations are calculated to reach a **target**
- E.g. we want character's hand to touch an object; IK calculates rotations of upper and lower arm to achieve this subject to constraints

The most common use for IK

Ragdolls

Ragdolls

- Attach a **rigid body** to each bone and run a **physics simulation**

Ragdolls

- Attach a **rigid body** to each bone and run a **physics simulation**
- Often used for death animations

Ragdolls

- Attach a **rigid body** to each bone and run a **physics simulation**
- Often used for death animations
- Many games mix and blend some or all of keyframe animation, IK, and physics simulation

Skinning

Skinning

- The character is animated by changing the bone transformations

Skinning

- The character is animated by changing the bone transformations
- **Skinning** is the process of applying these transformations to the vertices of the model

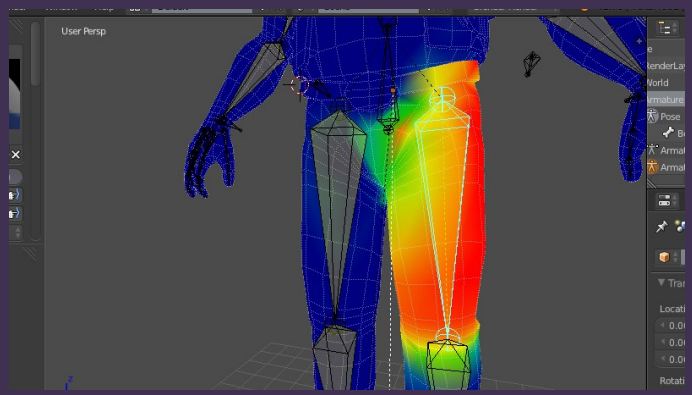
Skinning

- The character is animated by changing the bone transformations
- **Skinning** is the process of applying these transformations to the vertices of the model
- Generally handled by a **vertex shader**

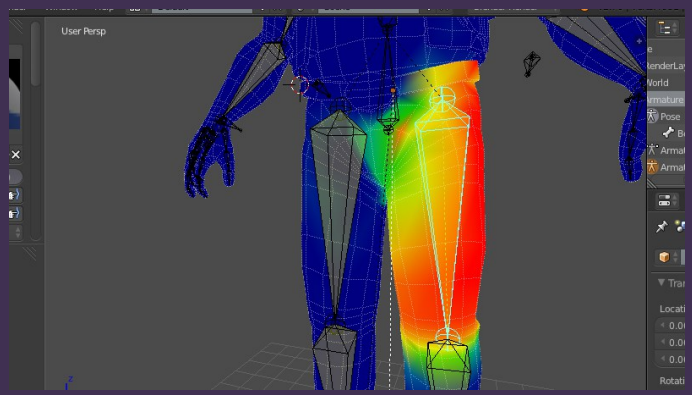
Skinning

- The character is animated by changing the bone transformations
- **Skinning** is the process of applying these transformations to the vertices of the model
- Generally handled by a **vertex shader**
- Uses **bone weights** to specify how much each vertex is affected by each bone's transformation

Bone weights

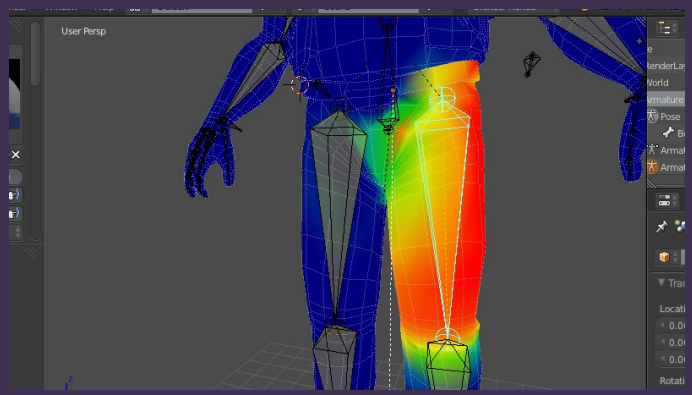


Bone weights



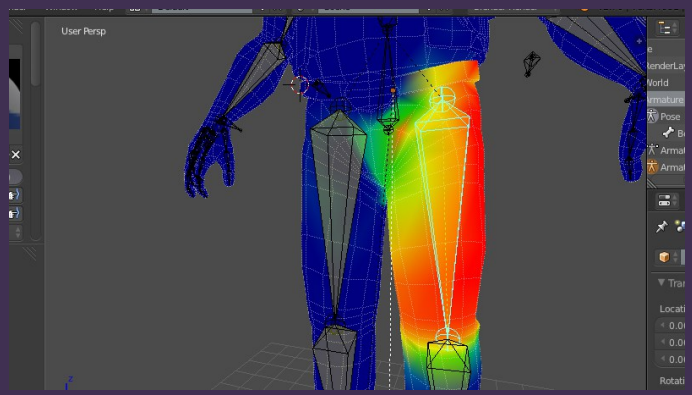
- Each vertex in the model has a list of **bone weights**

Bone weights



- Each vertex in the model has a list of **bone weights**
- Usually "painted" onto the model by the 3D artist

Bone weights



- Each vertex in the model has a list of **bone weights**
- Usually "painted" onto the model by the 3D artist
- Vertex position is calculated as a weighted average of joint transforms

Secondary effects

Secondary effects

- Additional movement that occurs as a result of the skeletal animation

Secondary effects

- Additional movement that occurs as a result of the skeletal animation
- For example: swinging clothes or hair, jiggle of body parts, skin sliding over bones

Secondary effects

- Additional movement that occurs as a result of the skeletal animation
- For example: swinging clothes or hair, jiggle of body parts, skin sliding over bones
- Can be **simulated** using a variety of techniques, though this is often expensive even for simplified methods

Secondary effects

- Additional movement that occurs as a result of the skeletal animation
- For example: swinging clothes or hair, jiggle of body parts, skin sliding over bones
- Can be **simulated** using a variety of techniques, though this is often expensive even for simplified methods
- A popular approach is to use **morph targets** (aka **blend shapes**)

Morph targets

Morph targets

- Take two (or more) **topologically identical** meshes (i.e. the same vertices in the same order)

Morph targets

- Take two (or more) **topologically identical** meshes (i.e. the same vertices in the same order)
- Deform each mesh to represent a different **target**, e.g. happy, sad, angry, neutral

Morph targets

- Take two (or more) **topologically identical** meshes (i.e. the same vertices in the same order)
- Deform each mesh to represent a different **target**, e.g. happy, sad, angry, neutral
- Create effects by **interpolating** the vertex positions to blend between the targets

Morph targets

- Take two (or more) **topologically identical** meshes (i.e. the same vertices in the same order)
- Deform each mesh to represent a different **target**, e.g. happy, sad, angry, neutral
- Create effects by **interpolating** the vertex positions to blend between the targets

Image source: `https://antongerdelan.net/opengl/blend_shapes.html`

Next steps

- **Review** the additional asynchronous material for more background on physics and animation techniques.
- **Attend** the workshop to learn how to use the Bullet Physics library in your own applications.