Working with Models

Computer Graphics - SET08116

EDINBURGH NAPIER UNIVERSITY

Outline

Triangles/Meshes

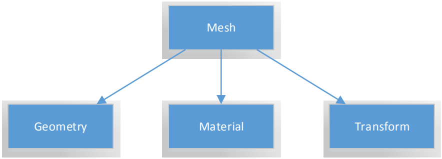
How do you currently **manage** your vertex/mesh data?

How do you think you'll **manage** your data and why?

What other things will you have in a scene you need to manage?

Meshes

- Up until now we have focused our understanding of working in 3D to what we have called a mesh.
- A **mesh is just a collection of data** that we use to represent a 3D object in a scene
- We have defined a mesh to have the following **3 properties**
  - **Geometry**
  - **Material**
  - **Transform**
- These provide different information to the renderer



Geometry

- This is where we started the module - we concerned ourselves with geometry
- Geometry is just the data that allows the object to be represented in the 3D scene
- This goes **beyond just the basic position data** used to represent the shape in 3D space
- We have also defined the **surface normals** to allow us to perform lighting calculations
- We also define data such as **texture coordinates** to texture our surfaces and colour data if necessary.
- Geometry data typically comes in vector format

Material

- The last few lectures on the rendering technology have focused on **visual appearance** - this is the material aspect of the mesh
- **Materials help us define what an object looks** like in the scene
- They are made up of various values, but typically we have a number of textures (one for texturing, another for normal mapping, etc.) and a collection of values that **describe how light interacts with the object**
- The values we have been using are diffuse reflection, specular reflection, shininess, etc.
- The **material is an important aspect or 3D rendering**, and probably the part you were unfamiliar with before now

Transforms

- Finally our mesh also has a transform - the data that describes **how the object is placed in the world**
- Previous weeks we have discussed the general mathematics behind matrix transformation
- This week we are looking far more at **concatenation of transformation** matrices to create **mesh hierarchies**
- Transforms in this manner are what drive our ability to work with complex models - we have already discussed the concept of coordinate spaces. Here we are just taking the idea a bit further

Linear Transforms

- Underpinning our work today is the **idea of a linear transformation**
- A linear transformation is just a **combination** of the three transforms we discussed in past weeks.
- For a linear transformation we have the following:

$$\mathbf{M} = \mathbf{TRS}$$

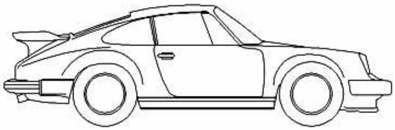
  where **M** is the model transformation matrix, **T** is the translation matrix, **R** is the rotation matrix, and **S** is the scale matrix.
- We are now going to look at how we can **combine model matrices to create complex 3D objects**

Heirarchical Models

- Up until now we have been dealing with our meshes as **single entities** made up of geometry, material and transform
- This is how the render framework has been built, so the discussion we are having **today** is either for information, or for those of you who want to attempt to **build something more complex in your final scene**
- What we want to do now is understand how we can **combine different pieces of mesh together** to form more complex objects
- To do this, we use a technique called **modelling hierarchies**
- Modelling hierarchies allow us to think about a complex 3D model as its component parts

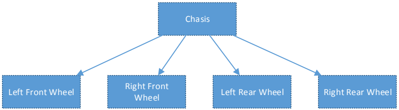
Car Example

- Let us begin with a **simple car example**
- A car can be considered to be made up of a number of **different parts**, but here we will only consider the most basic outside parts. These are
  - The body or chassis
  - The four wheels
- Our job is to think about how we can represent these objects to that we **treat the car as one object, while at the same time be able to work with the parts independently** (e.g. turn the wheels without turning the car)



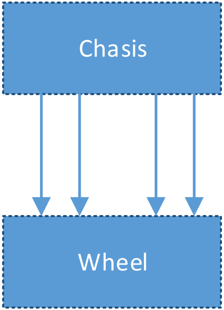
Trees

- If we think about a car we can start to understand what we must do
- The **wheels themselves can rotate (turn) independently**, but they **must be connected to the car itself**
- Therefore, the position of the wheels depend on the position of the car
- If the **body of the car moves, then the position of the wheels also changes**
- We can illustrate this dependency by drawing a hierarchy of the relationships



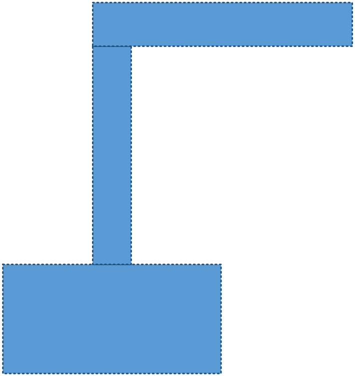
Reusing Geometry

- We can also understand the car as the geometry involved
- The **car only needs two pieces of geometry**
  - The body
  - The wheel
- **Reusing the geometry** in this manner is useful, but can be confusing for our current understanding
- We will return to geometry reuse in a **couple of weeks when we discuss scene graphs**



Robot Arm Definition

- Let us now look at a bigger example - a robot arm
- A robot arm can be defined as **three separate pieces**
  - The base
  - The lower arm
  - The upper arm
- Each of these pieces can **move independently of one another** - we just need to understand how they do this



Base Transform

- The base of the robot arm can be considered as the **root of the model** - it is the one which **dictates the actual positioning** of the object
- As such, the base has a position (translation) into the world
- The base can also rotate on its Y-axis, giving it one degree of rotational freedom

Lower Arm Transform

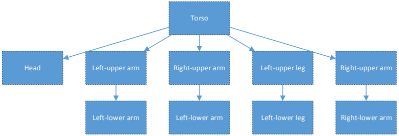
- The **lower arm** is connected to the base, and therefore its **position is determined by the base's position**
- The lower arm is also able to rotate at its joint with the base. It can do this on the Z-axis
- The lower arm has an offset from the base. Therefore, we can rotate the lower arm as required, and then offset it by the base's position

Upper Arm Transform

- The upper-arm is **connected** to the lower-arm at a **joint**
- The upper-arm therefore has its position determined by the lower-arm's position
- The upper-arm can also rotate on the Z-axis at the joint
- The upper-arm is therefore offset by the lower-arm, which itself is offset by the base. We rotate the upper-arm as desired, then take into account the lower-arm offset, before finally offsetting by the base.

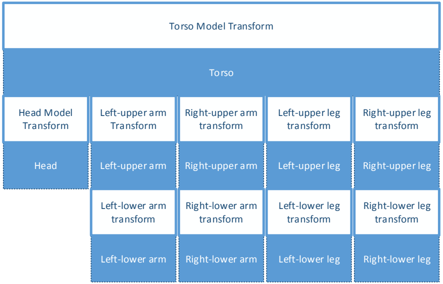
Trees and Traveral

- When it comes to representing our data, we can take a couple of different approaches
- The main idea is that we have a **tree like data structure** which we can use to store data
- The **tree has a root node, and a number of child nodes**
- Each **child node is affected by the node above** it in the hierarchy, which in turn is affected by the nodes above it
- We need a method to allow simple processing of this information



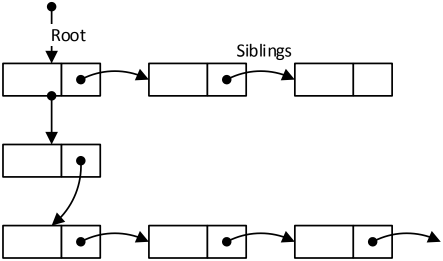
Matrix Stacks

- One way we could process the data is in the use of **matrix stacks**
- A matrix stack allows us to **combine transforms together**, while still being able to go back to access a previous matrix
- For example, we could **render the body, push the body transform matrix, multiply by the head transform**, render the head, pop the body transform matrix back and then multiply by the upper-arm transform, etc.



Tree Data Structures

- A **better method** is to use a **tree data structure** and just traverse down the tree
- We **render the root, then pass it's transform down to a child node**
- The child node renders itself, and passes the transform to its child
- And so on down each branch of the tree



Tree Traversal

- Understanding **tree traversal is a very important aspect of computation** in general - but especially so in graphics rendering
- We use **trees to represent models**
- We use **trees to represent scenes** (more on scene graphs later)
- We can also use **trees to represent shader effects** and render passes
- You should be or should be about to cover trees in the algorithms and data structures module - you should get used to them now

Animation

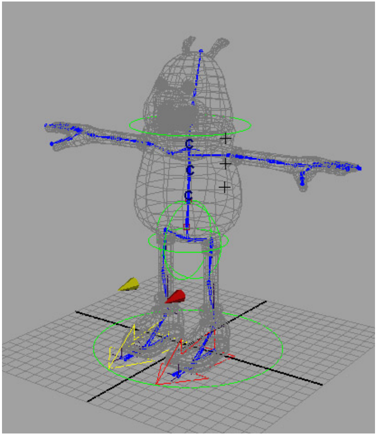
- Now that we know how to treat parts of our model independently, we can start discussing animation of models
- We are going to take a very trivial view of animation in that we will just look at the basic concepts
- If you are interested in model animation, ‘AssImp’ library, which can help you.
- For the moment, let us look at some of the methods of describing animation

Key Frame Animation

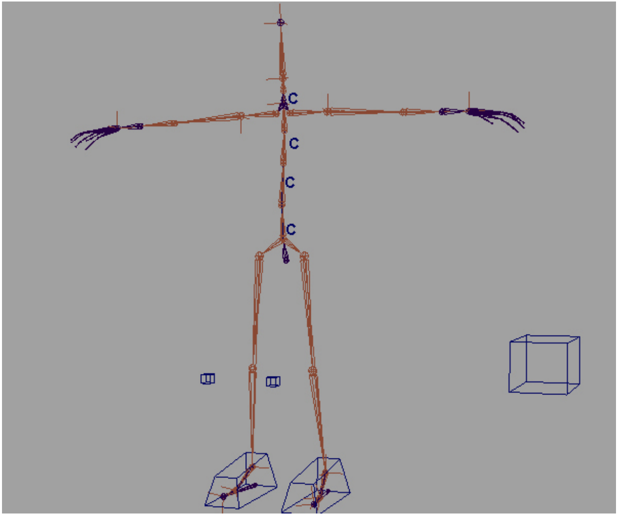
- The first method we will discuss is **keyframe animation**
- Keyframe animation works on the principle of having a **number of keyframes defined in our data** - think of these as particular **positions of vertices at particular points in time**
- Our job is to **interpolate** between the different keyframes based on the time passed - thus giving the parts between the frames
- Keyframe animation is very useful when using motion capture

Rigging

- 3D modellers use a technique known as **rigging** to enable animation of 3D models
- Rigging involves our 3D data being attached to a **rig (like a skeleton)** which can be manipulated to animate separate parts of the object
- Each **bone in the rig is attached to a collection of vertices**
- **Moving the bone moves the vertices**, thus providing the ability to manipulate the individual parts of the model



Bones



Kinematics

- **Kinematics** is a method that uses physical calculations to determine the position of bones
- The idea is that we can work out the **orientation and position of individual model parts** based on the physical forces places upon them, thus allowing a more realistic representation of **movement over time**
- In fact, we typically use inverse kinematics to work out our movement
- **Inverse kinematics is where we know the position we want our object to be and how long it takes to be there**. From that we work backwards with our phyiscal calculations

Vertex Blending

- Rigging and animation from rigging can actually lead to strange effects
- What can happen is that we end up with the individual parts of a model looking just like that - **individual parts**
- In a real body, different bones can affect the position of an individual part
- To overcome this we **allow vertices to be manipulated by more than one bone** using a technique called **vertex blending**
- Real-Time Rendering contains more information for the interested

Summary

- You should now have a grasp on how we go about **representing objects** to allow us to treat the parts individually
- You should also now have an understanding of how we can **manipulate the transforms of an object to move the parts individually** - we are just advancing our coordinate spaces idea
- You should also have an idea of how we can **represent the data in a 3D model** so we can move the parts independently
- Finally, we have covered animation at a very high level - it is up to you to explore further if you are interested

Recommended Reading

Chapter 8 of Interactive Computer Graphics provides more detail of the concepts we have discussed today.