Scene Hierarchies

Last Time: Composition

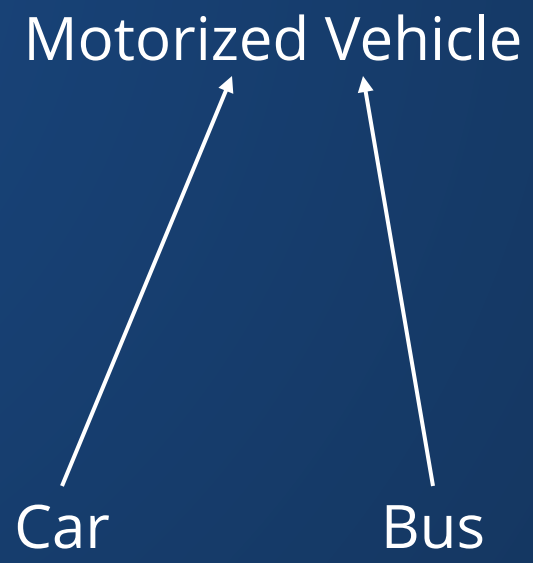
If objects consist of multiple other objects, we can use *composition* (*has-a*) to simplify programs and make them easier to reason about.

```
1  class Bicycle{
2    Frame frame;
3    Wheel frontW;
4    Wheel backW;
5    float x, y;
6    float wheelDist;
7
8    Bicycle(Frame f, Wheel w){  }
9
10   void displayBike(){ }
11
12   void moveBike(float dx){  }
13 }
```

Last Time: Inheritance

If some classes (objects) are more specific instances of other classes (objects), we can use *inheritance* to model this. This is an example of an *is-a* relation

```
1  class MotorizedVehicle {
2    float fuel, speed;
3    void addFuel(float amount){ /* */ }
4    void accelerate(float rate){ /* */ }
5  }
6
7  class Car extends MotorizedVehicle {
8    color color;
9  }
10
11 class Bus extends MotorizedVehicle {
12   int numSeats;
13 }
```



Recap: Rules of Inheritance

- Use inheritance only if your derived class can be thought of as a more specific instance of the base class.
- Child classes retain all the members (fields + methods) of their parent, and can add more.
- Use `extends` keyword to declare a subclass relation.
- Child class constructors should call `super()` to access their parent constructor.
- Methods that are redefined in the child *override* the ones in the parent.

Questions

Why was there another Hands-On due yesterday?

Due dates for hands-on activities are optimistic: they assume we're covering as much material in class as possible.

The actual due date for the hands-on will be based on when we get to it in class.

Why bother with subclassing if you're just going to override the methods anyways?

```
1 class MotorVehicle {
2   // blah blah blah
3   void accelerate() {  }
4 }
5
6 class Car extends MotorVehicle {
7
8 }
```

```
1 void makeThingGoFast(MotorVehicle v){
2   v.accelerate();
3   v.accelerate();
4   v.accelerate();
5   v.accelerate();
6   v.accelerate();
7   v.accelerate();
8 }
```

What relation between the two classes guarantees that `v` has an `accelerate()` method?

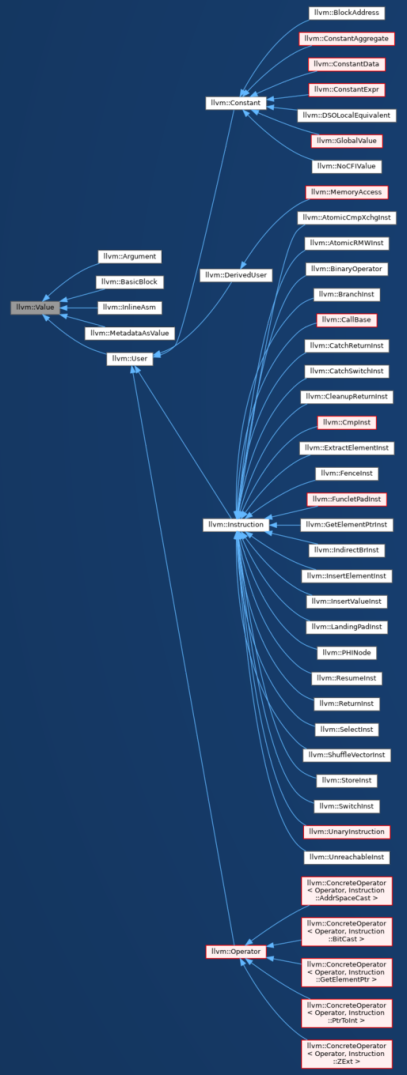
Why bother with subclassing if you're just going to override the methods anyways?

```
1  class MotorVehicle {
2    void accelerate() {  }
3  }
4
5  class Car {
6    void accelerate() {  }
7  }
8
9  class CS324E {
10   void accelerate() {  }
11 }
12
13 class ProjectSchedule {
14   void accelerate() {  }
15 }
16
17 class ParticleBeam {
18   void accelerate() {  }
19 }
```

Which of these accelerate() methods do similar things?

Inheritance tells us that related objects are likely to behave similarly.

How many classes do you usually see in large programs?



Quite a few, as it turns out! It takes a while to learn the object hierarchy in these projects, but it makes it easier in the long run.

A few notes about this:

- In Java (and Processing), each class can only have one parent.
- Great pains have been taken here to have a class hierarchy that makes sense (i.e. no `Cars` that are actually `ParkingGarages`). If the hierarchy were nonsense, it would make things much harder to understand.

Can you explain a little more about `super`?

`this` can be used as a variable to access the current object.

```
1 class Cow {
2   void moo(){
3     println(this.name + " says Moo");
4   }
5 }
```

`this` can be used to call methods of the current class.

```
1 class Cow {
2   void beHappy(){
3     this.eatGrass(field);
4     this.moo();
5   }
6 }
```

`this` can be used to call constructors of the current class.

```
1 class Cow {
2   Cow(String name){
3     this.name = name;
4   }
5   Cow(){
6     this("Peanut");
7   }
8 }
```

These last two uses are shared by `super`, but refer to things in the superclass!

`super` can be used to call methods of the *superclass*.

```
1 class SuperCow extends Cow {
2   void beHappy(){
3     this.fightCrime();
4     // Would infinite recurse if it was
5     // this.beHappy();
6     super.beHappy();
7   }
8 }
```

`super` can be used to call other constructors *of the superclass*.

```
1 class SuperCow extends Cow {
2   SuperCow(String name, String power){
3     super(name);  // Must exist!
4     this.power = power;
5   }
6 }
```

Why does super() have to be the first thing in a constructor if it's present?

Java said so.

There are explanations out there that you can find, but they're a little complex.

It boils down to:

- Java was trying to enforce some rules about object validity
- Forcing you to call super() first is one way to try to enforce that
- This doesn't really accomplish what it was supposed to
- There were probably other ways of enforcing these rules

Why bother with super()?

```
1  class BaseClass{
2    int val1, val2;
3    BaseClass(int x, int y){
4      val1 = x;
5      val2 = y;
6    }
7    BaseClass(){
8      val1 = 0;
9      val2 = 5;
10   }
11 }
12
13 class Sub1 extends BaseClass {
14   Sub1(int x, int y){
15     val1 = x;
16     val2 = y;
17   }
18   Sub1(){
19     val1 = 2;
20     val2 = 5;
21   }
22 }
```

```
1  class Sub2 extends Sub1 {
2    int val3;
3    Sub2(int x, int y, int z){
4      val1 = x;
5      val2 = y;
6      val3 = z;
7    }
8    Sub2(){
9      val1 = -1;
10     val2 = 5;
11     val3 = 4;
12   }
13 }
```

Uh-oh. The requirements changed! Now we need to guarantee that the values are all between 0 and 4. In the constructor, if the value is outside the range, we clamp it.

Why bother with super()?

```
1  class BaseClass{
2    int val1, val2;
3    BaseClass(int x, int y){
4      val1 = x;
5      val2 = y;
6    }
7    BaseClass(){
8      this(0, 5);
9    }
10 }
11
12 class Sub1 extends BaseClass {
13   Sub1(int x, int y){
14     super(x, y);
15   }
16   Sub1(){
17     this(2,5)
18   }
19 }
```

```
1  class Sub2 extends Sub1 {
2    int val3;
3    Sub2(int x, int y, int z){
4      super(x, y);
5      val3 = z;
6    }
7    Sub2(){
8      this(-1, 5, 4);
9    }
10 }
```

Now how many places do we need to change?

```
1  class BaseClass{
2    int val1, val2;
3    BaseClass(int x, int y){
4      val1 = x;
5      val2 = y;
6    }
7    BaseClass(){
8      this(0, 5);
9    }
10 }
11
12 class Sub1 extends BaseClass {
13   Sub1(int x, int y){
14     super(x, y);
15   }
16   Sub1(){
17     this(2, 5);
18   }
19 }
```

What happens if my class hierarchy gets too complex?

But this does touch on one of the distinct weaknesses of this type of programming.

Code Review

How can I control the number of members that the child inherits from the parent?

You can't.

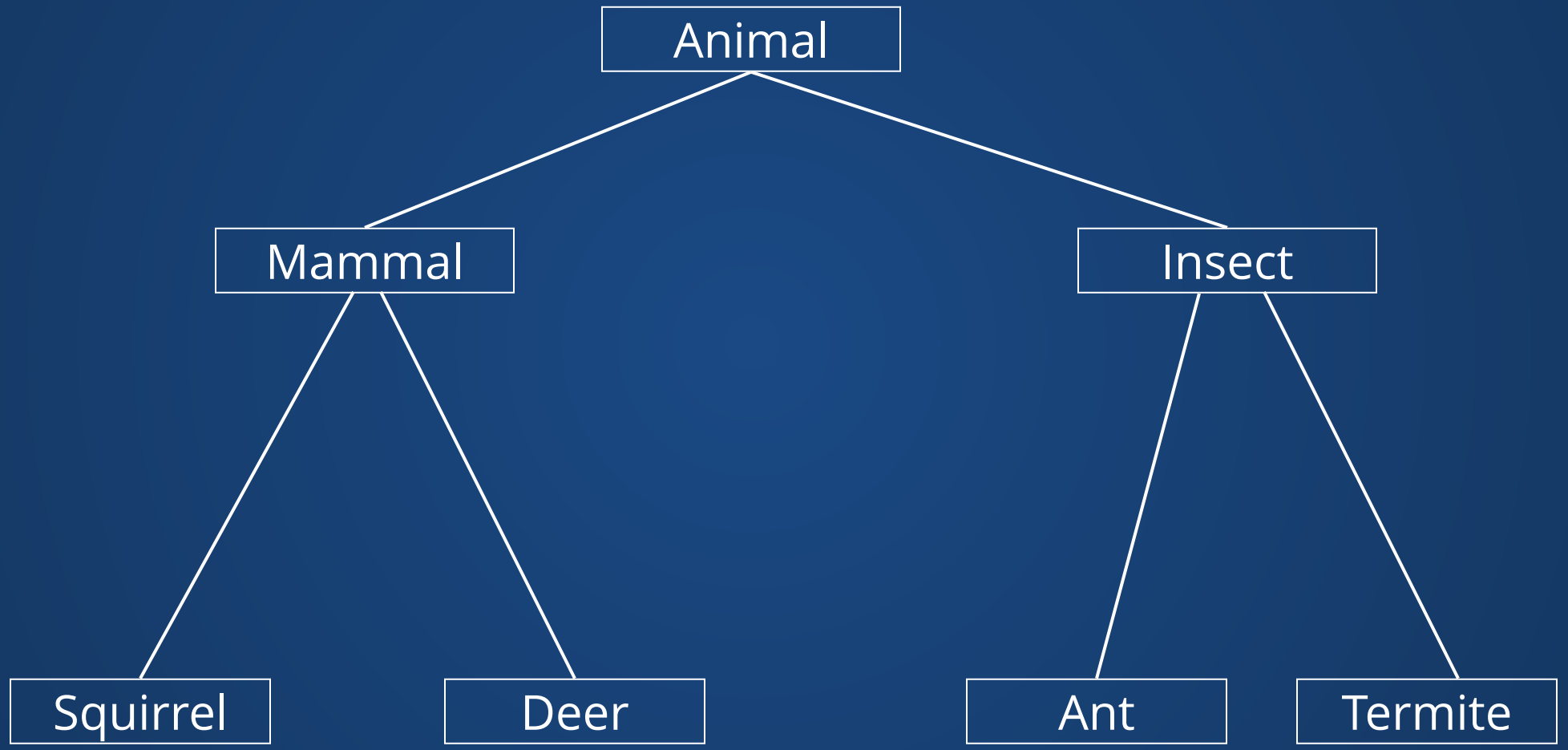
By saying your child is a special type of the parent, you're saying **the child can do everything the parent can and more**.

Deleting members or preventing the child from inheriting them would fundamentally break this guarantee.

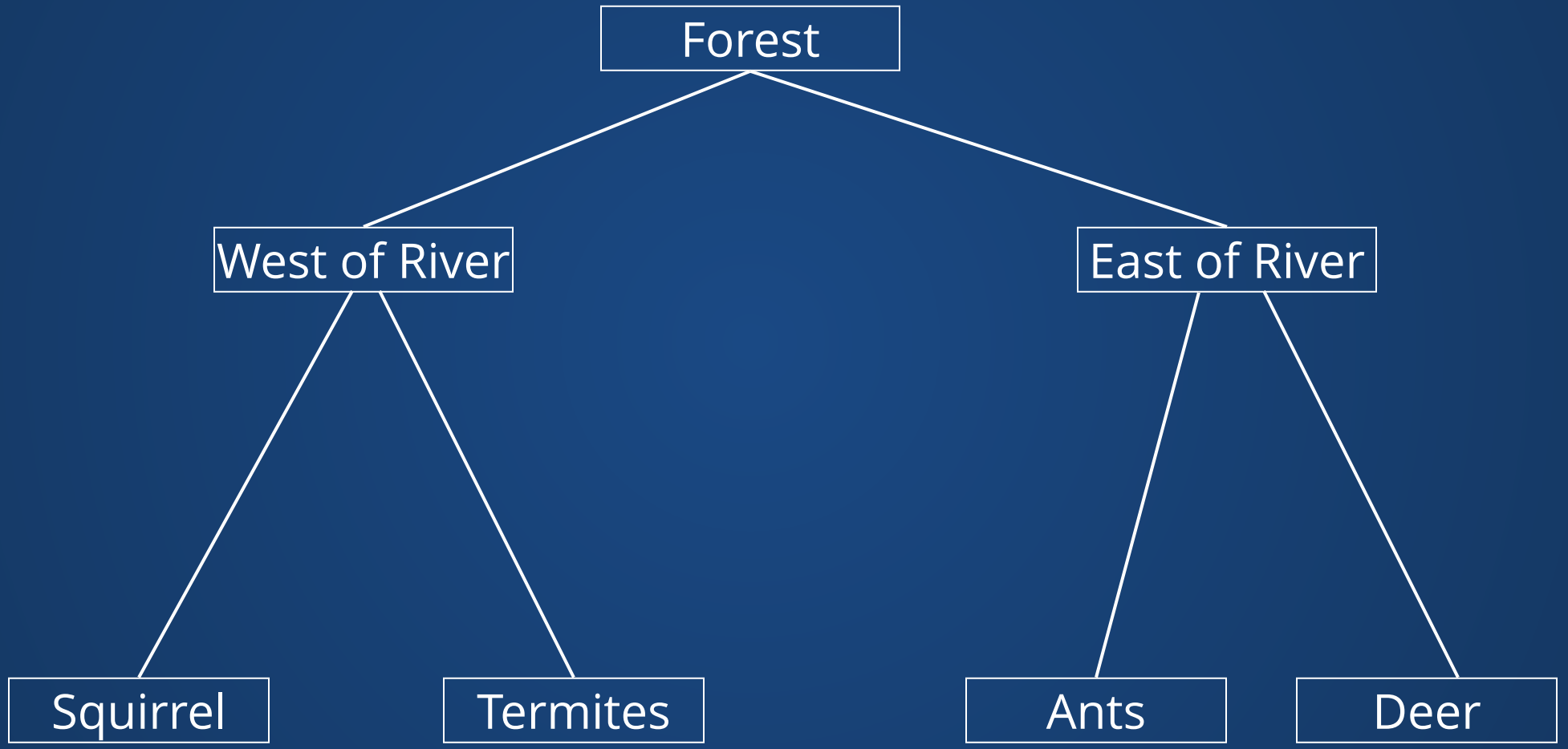
```
1  class MotorVehicle {
2    void accelerate() {  }
3  }
4
5  class Car extends MotorVehicle{
6    // Cast magical spell to prevent
7    // method from being inherited
8    // ABLOOGY WOOGY WOO
9    // void accelerate() {  }
10 }
```

Today: Other Hierarchies

Hierarchy: Taxonomic



Hierarchy: Location



Different organization, same entities!

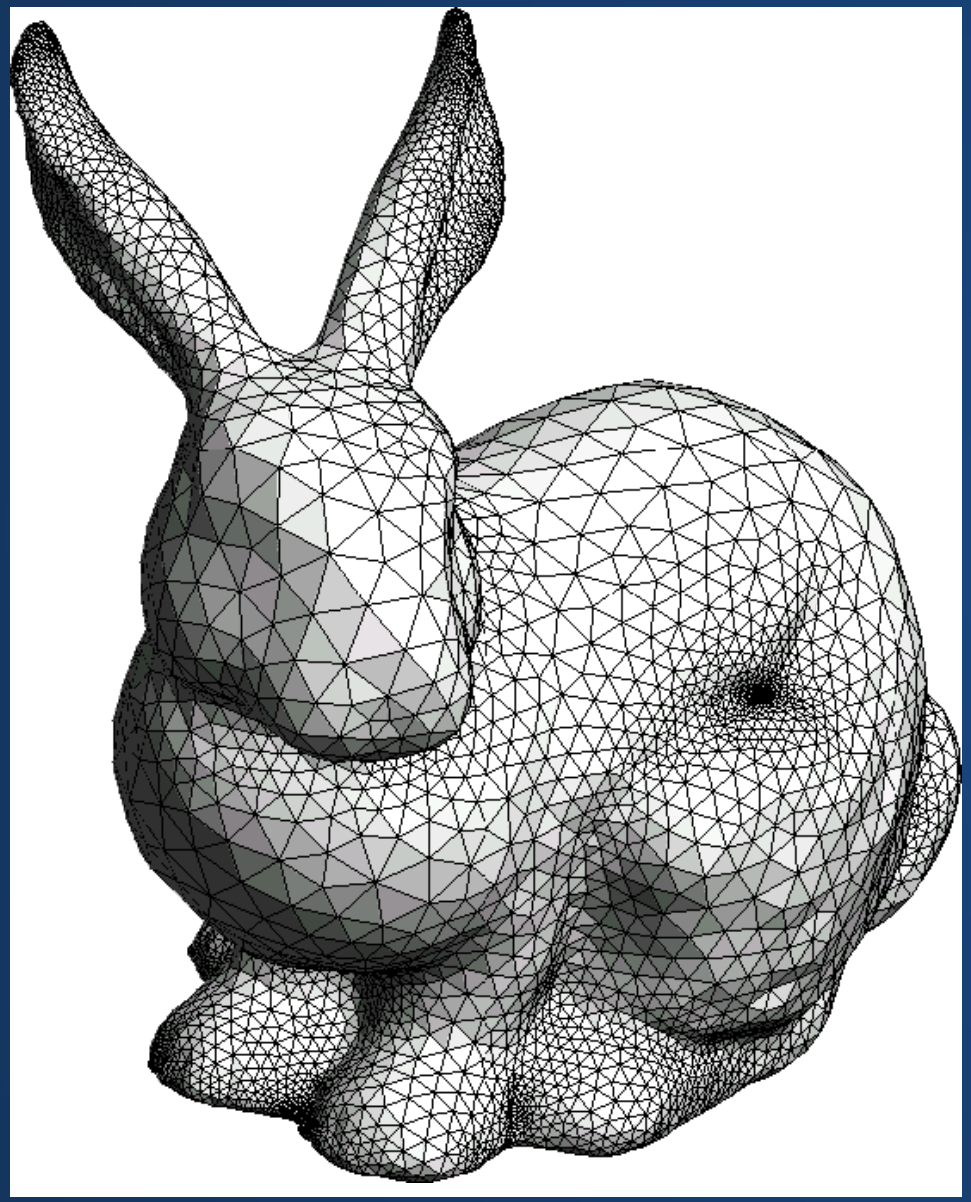
We can organize things hierarchically using different criteria, which naturally leads to different organizations.

Today we're going to talk about *spatial* organization.

Shapes

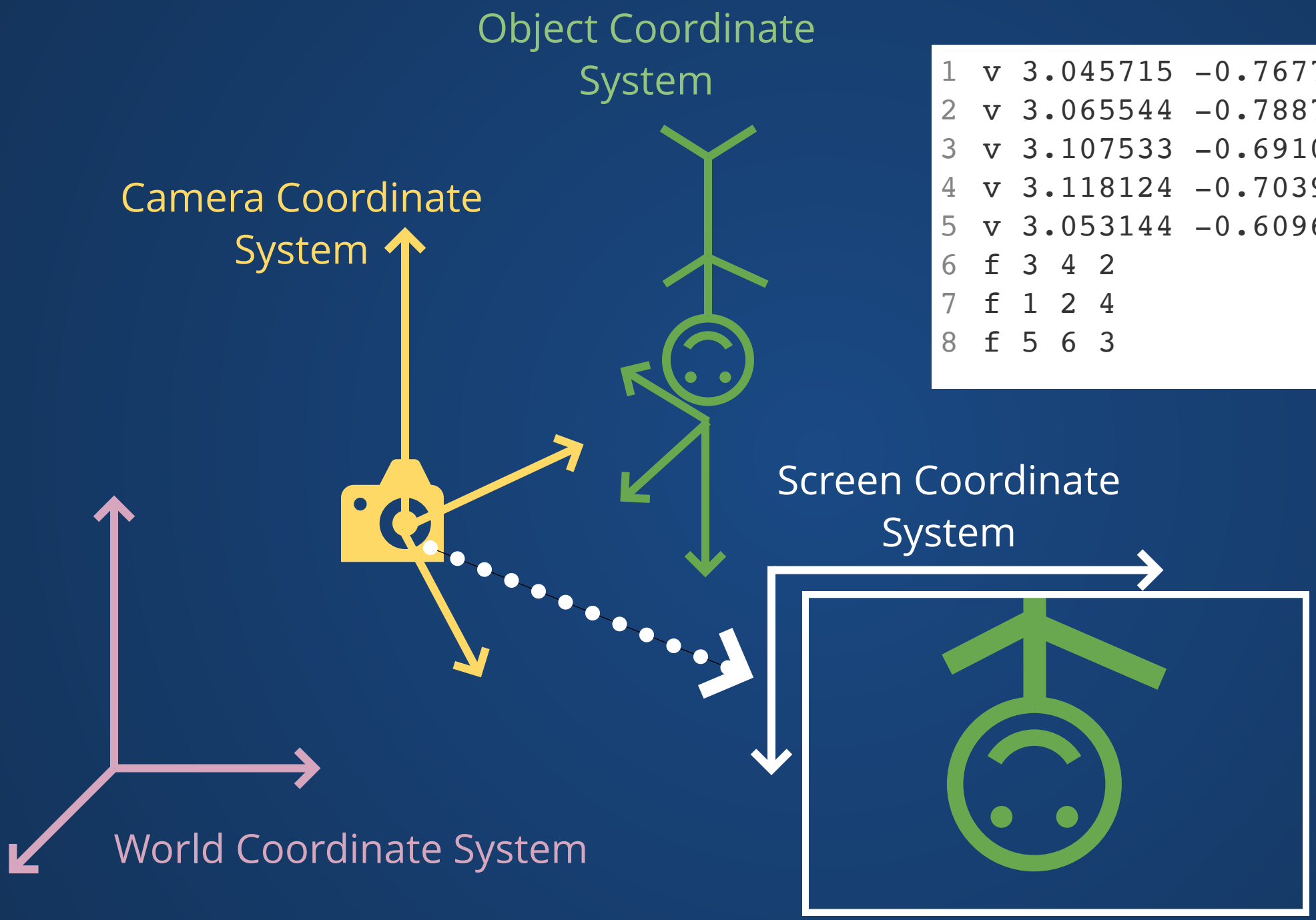
An Aside

How do we define shapes?



1. Define a set of points in space.
2. Connect the points to form edges.
3. Combine the edges to form faces.
4. Combine the faces to form meshes.

```
1  v 3.045715 -0.767786 -0.261741
2  v 3.065544 -0.788768 0.000000
3  v 3.107533 -0.691019 -0.264602
4  v 3.118124 -0.703925 0.000000
5  v 3.053144 -0.609681 -0.326931
6  f 3 4 2
7  f 1 2 4
8  f 5 6 3
```




```
1  v 3.045715 -0.767786 -0.261741
2  v 3.065544 -0.788768 0.000000
3  v 3.107533 -0.691019 -0.264602
4  v 3.118124 -0.703925 0.000000
5  v 3.053144 -0.609681 -0.326931
6  f 3 4 2
7  f 1 2 4
8  f 5 6 3
```

Vertices

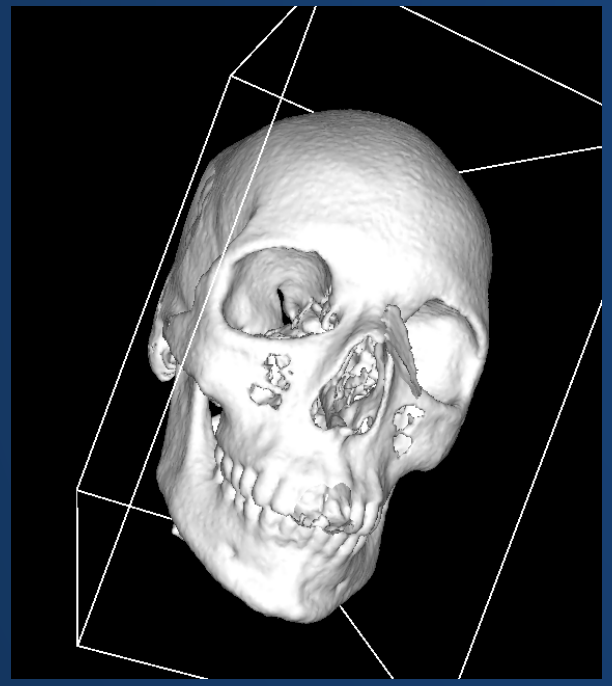
A *vertex* is a point that provides geometric information (and is the corner of a mesh).

Multiple vertices define a *polygon* or shape.

Conventionally, vertex data is given in the **world space** as opposed to the screen space.

Polygons

The default representation for objects in graphics.
In some cases, we even use only triangles---this is often called a "trimesh".



Scenes

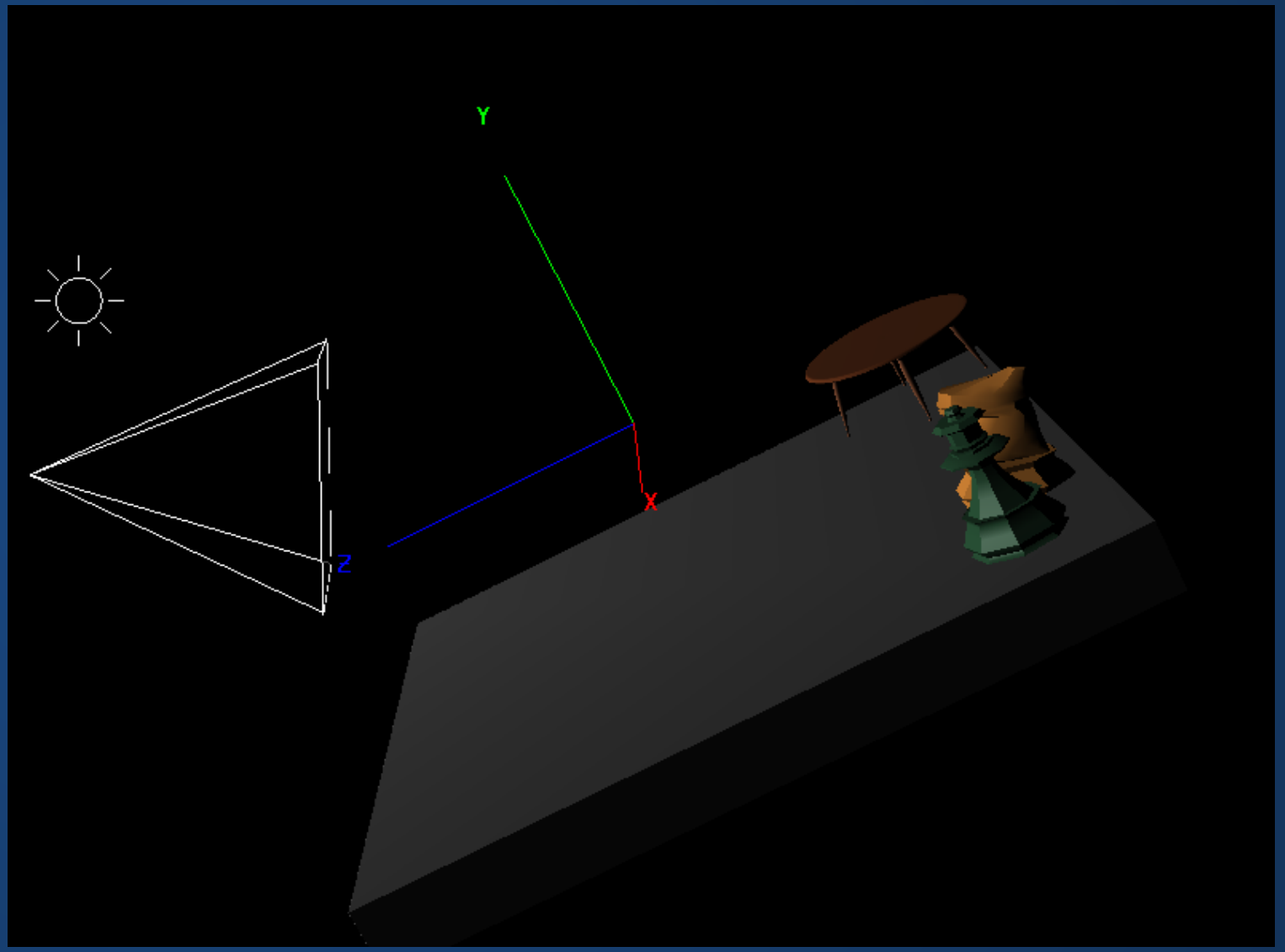
What is a Scene?

- A space we want to *render* (draw) on our screen
- Can be 2D or 3D
- What can a scene include?
  - Objects
  - Lights
  - Camera
- What types of scenes have we drawn so far in this class?

Scenes in Animation

Often like a movie set:

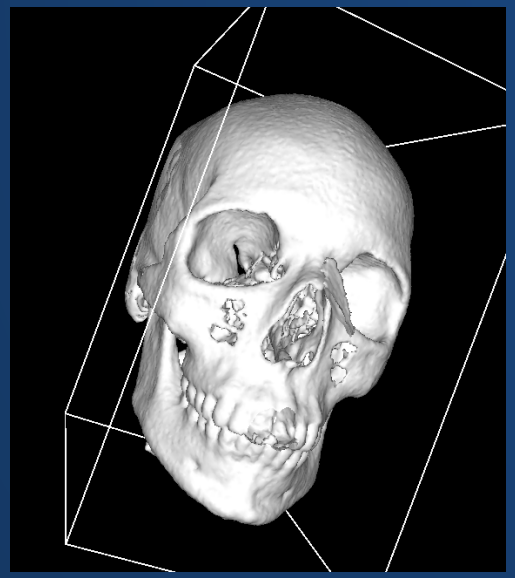
- Agents (actors)
  - Scripted
  - Player-Controlled
- Props for interaction
- Lights for shading
- Camera for rendering

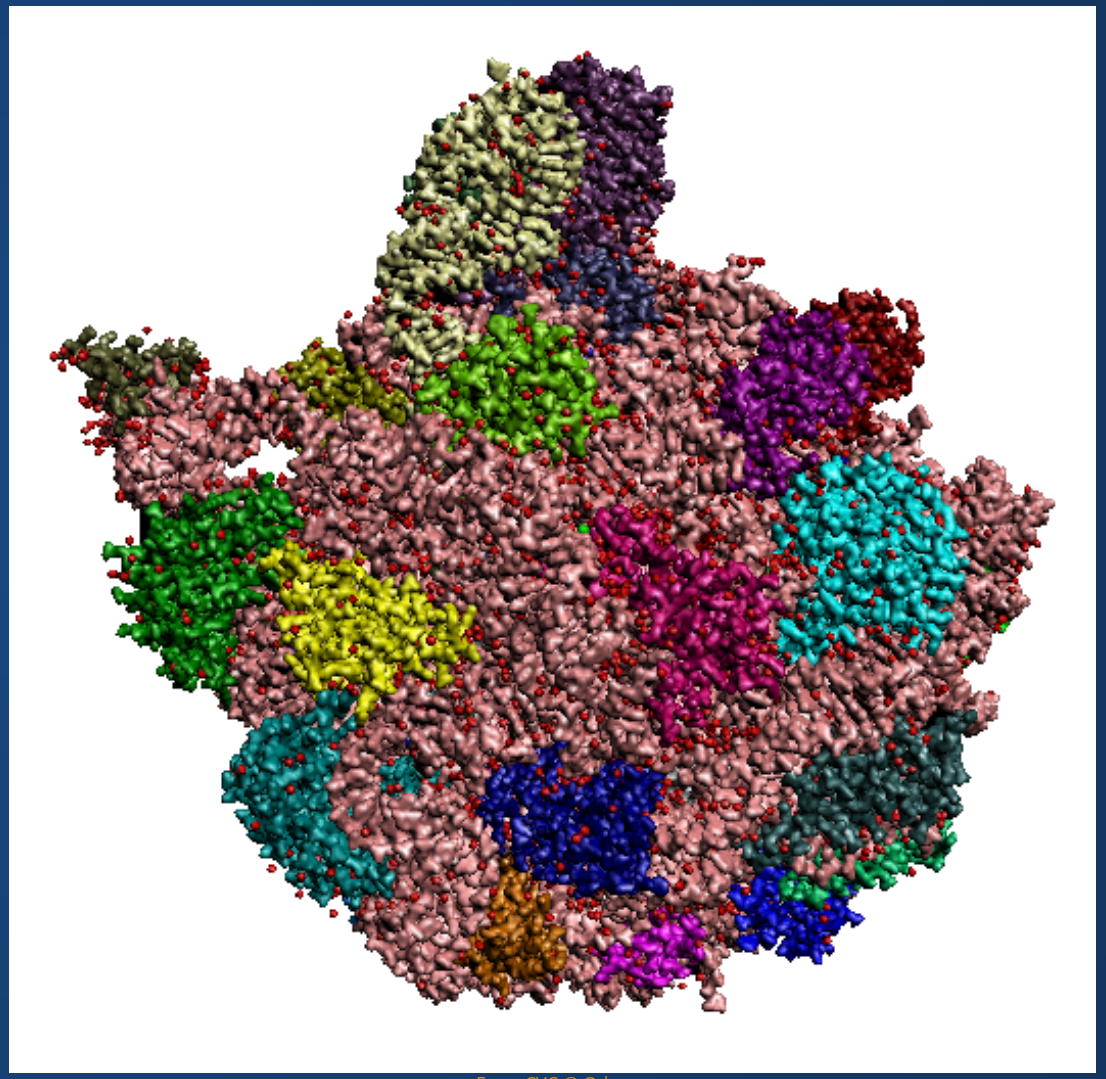


Scenes in Visualization

Surprisingly similar to animation/games!

Not something often thought about, even by people who make these visualizations.





From CVC @ Oden

Scene Hierarchies

Organizing Things Is Good!

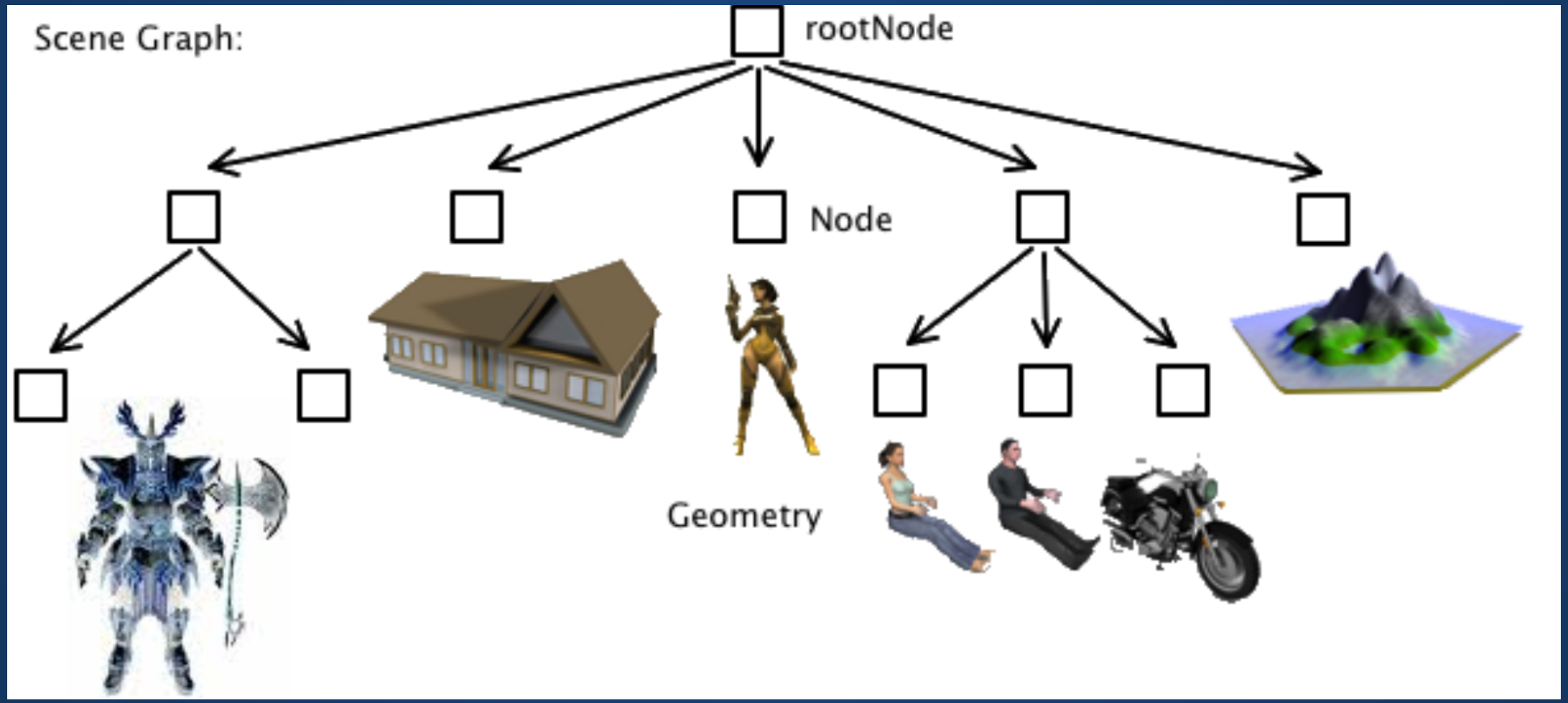
How do polygons relate to each other?

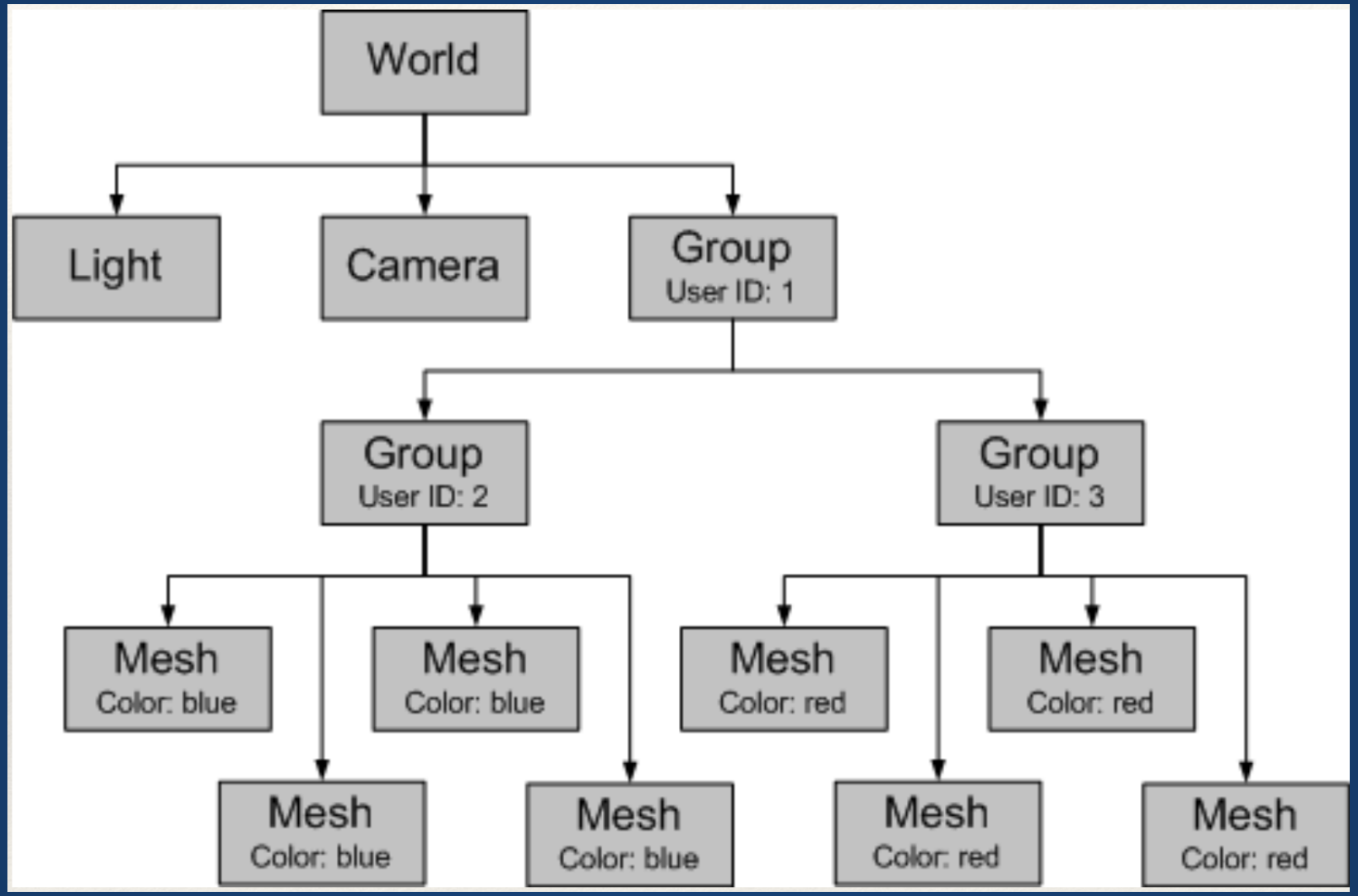
How do objects relate to each other?

https://www.youtube.com/embed/-XLBhlSxwX8?enablejsapi=1

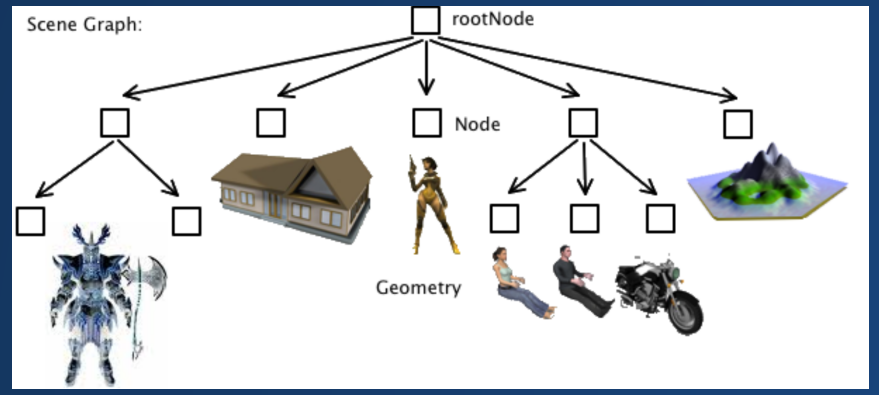
Scene Graphs

Tree hierarchy representing the relationship between objects in a scene.




World
Light
Camera
Group
User ID: 1
Group
User ID: 2
Group
User ID: 3
Mesh
Color: blue
Mesh
Color: blue
Mesh
Color: red
Mesh
Color: red
Mesh
Color: blue
Mesh
Color: blue
Mesh
Color: red
Mesh
Color: red

Scene Hierarchies *are not* OOP!



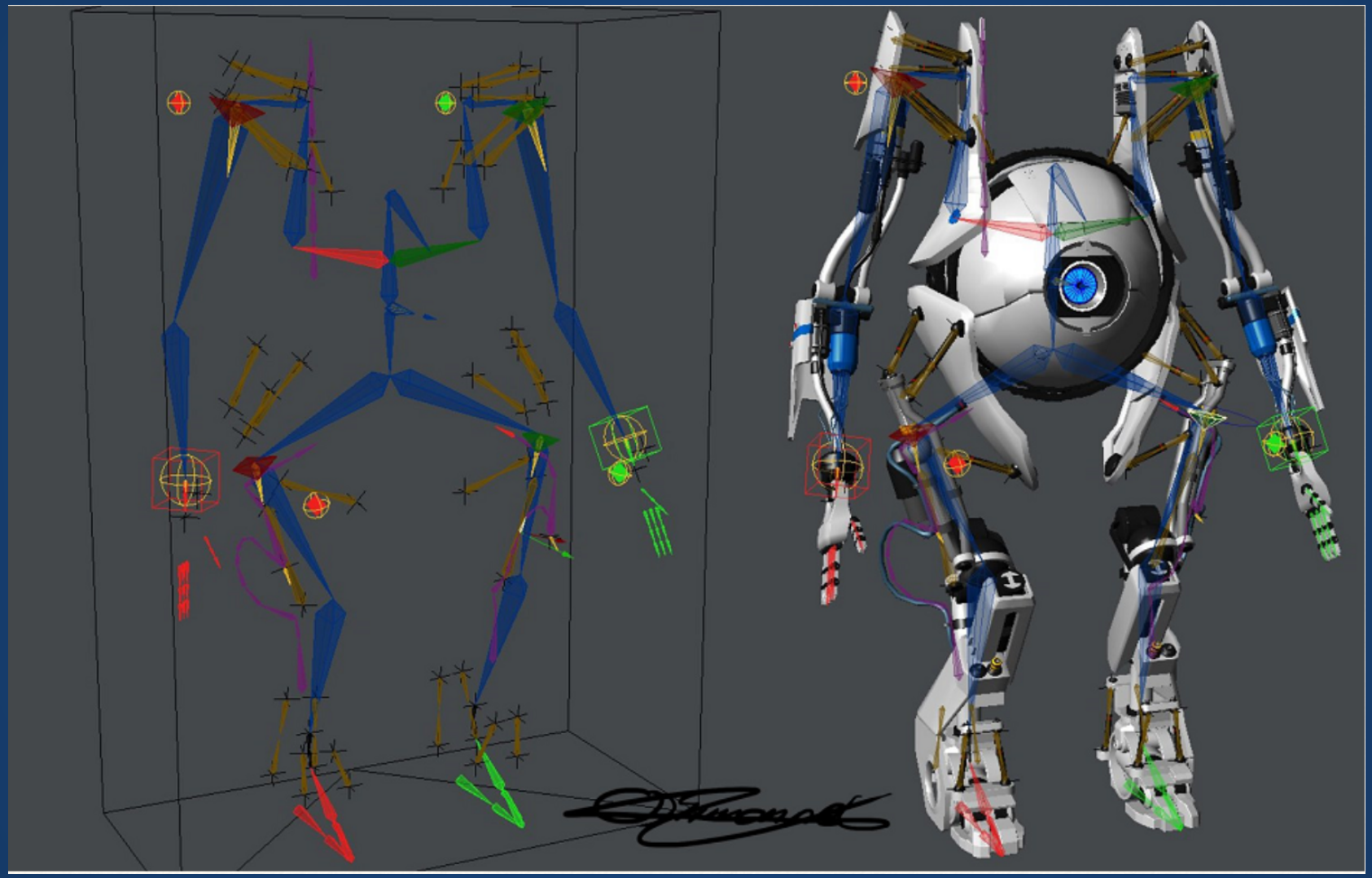
Looks a bit like an inheritance graph, but it is not!

People and motorcycles are not instances of the same class (well, maybe a generic "Object" class).

But we can use composition to model these sorts of things. For example, we can have a `Transform` which tells us how to transform this particular node.

Scene Graphs

Modeling and Animation



If we were moving each vertex of Atlas individually, how many different movements would we need to apply?

But joint movements are related to each other!

Example: Bending the elbow/shoulder changes the positions of the hand.

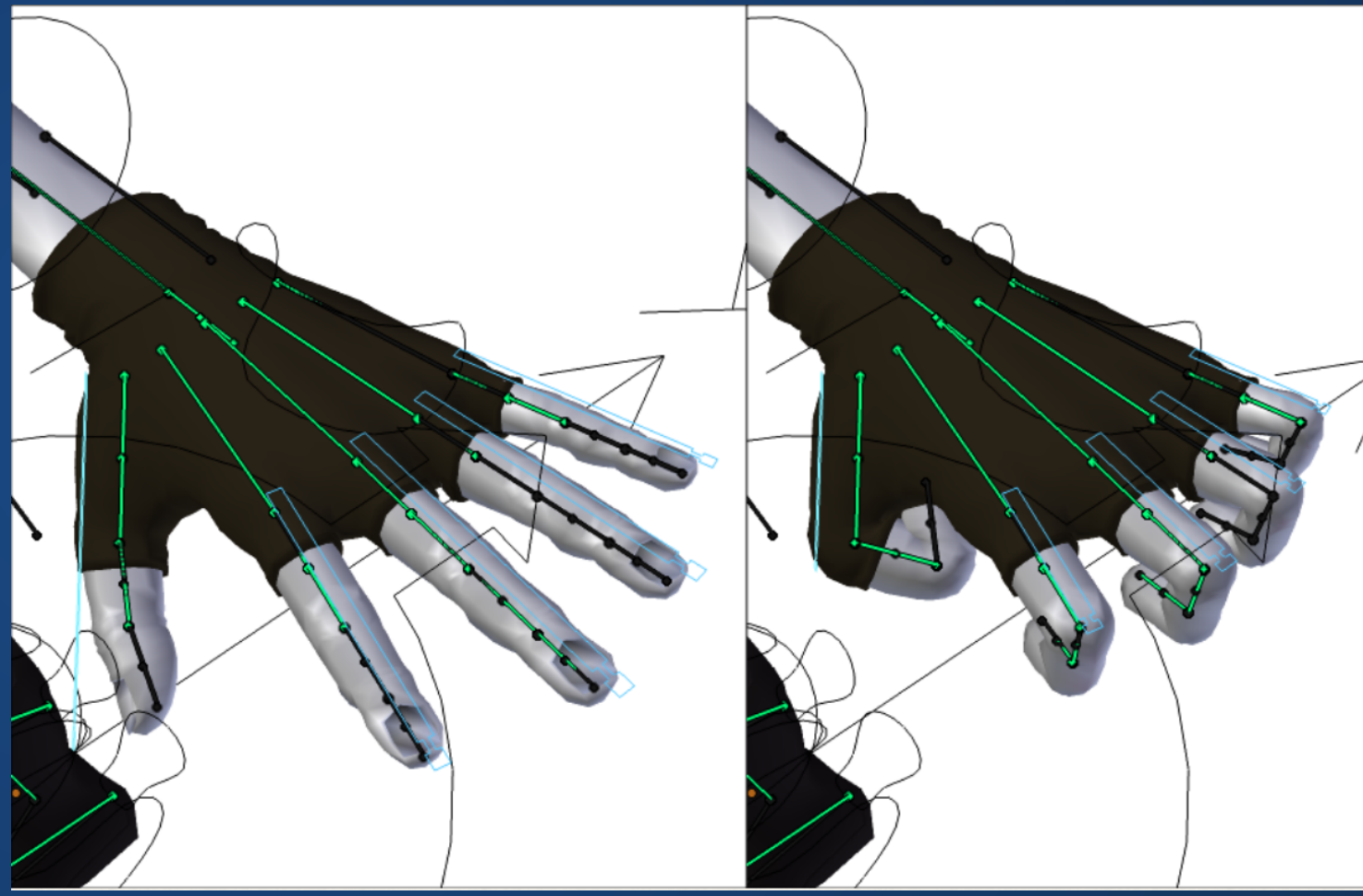
Hierarchical structure avoids having to move each vertex individually.

The hierarchy is based on the object design, not haphazard or random.

Consider the Pixar lamp: what is a hierarchical model that captures its degrees of motion?

Modeling to Animation

- *Modeling*: Set shape and form
- *Rigging*: Set underlying bone structure
- *Skinning*: Map skin (surface) onto bones
- *Animating*: Position bones to move the shape



Animation Hierarchy Examples

Hands-On: Scene Hierarchy

0. Read the Project 2 spec if you haven't yet.

1. Design a scene hierarchy for your project 2.

2. Design the individual objects and how each object will have two levels of animation.

Submit a .txt file describing your plans for the project.

Index Cards!

1. Your name and EID.

2. One thing that you learned from class today. You are allowed to say "nothing" if you didn't learn anything.

3. One question you have about something covered in class today. You *may not* respond "nothing".

4. (Optional) Any other comments/questions/thoughts about today's class.