

# Game Development I

Panda 3D, Project 3

# Tips

- Clean hierarchy, cleared history
- use `pview` to test model with all animations before handing off to programmers. Inverted normals are *very* common (`pview bob-model.egg bob-anim.egg`)
- Break up large geometry into separate nodes – there are lighting restrictions
- Any part of a model that’s going to be treated differently than another (e.g. a semi-transparent component) should have its own node
- Use the “Flagging Objects from Maya” script for collisions



```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```



Rensselaer

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

input file



Rensselaer

coordinate  
system

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

input file



Rensselaer

coordinate  
system

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways



Rensselaer

coordinate  
system

animation  
mode

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways



Rensselaer

coordinate  
system

animation  
mode

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

model = mesh + rig  
chan = animation  
channel

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways



Rensselaer

coordinate  
system

animation  
mode

character  
name

```
maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb
```

model = mesh + rig  
chan = animation  
channel

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways



Rensselaer

coordinate  
system

animation  
mode

character  
name

maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb

model = mesh + rig  
chan = animation  
channel

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways

tells Panda which  
animations are associated  
with a particular mesh



Rensselaer

coordinate  
system

animation  
mode

character  
name

output  
file

maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb

model = mesh + rig  
chan = animation  
channel

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways

tells Panda which  
animations are associated  
with a particular mesh



Rensselaer

coordinate  
system

animation  
mode

character  
name

output  
file

maya2egg2012 -cs y-up -a model -cn bob -o bobmodel.egg bobmodel.mb

model = mesh + rig  
chan = animation  
channel

output file name,  
called from code

input file

Panda uses Z-up, so  
this prevents the  
model from  
importing sideways

tells Panda which  
animations are associated  
with a particular mesh



Rensselaer

# Special Map Hypershade Inputs

all maps must be applied to a phong shader

texture → color

normal → normalCamera

opacity (white = opaque) → reverser → transparency

gloss → specularColor

glow → incandescence (need to run `setBloom()` on a Filters object)



Rensselaer

# Voodoo Global Variables

This is a special import statement which sets-up a bunch of shortcut variables to make life easier for you.

```
import direct.directbase.DirectStart
```

DirectStart loads most of the other Panda3D modules, and causes the 3D window to appear.

Makes an instance of showbase

base

Use these global **NodePath** reference names

camera

Short for base.camera

base.cam  
default camera

base.camLens  
default lens

taskMgr

run()  
launches

+ render

+ render2d

camera2d

camera for render2d

+ aspect2d

Is actually a child of render2d

See source: [panda3d/direct/showbase/ShowBase.py](#)

# Types of Panda Nodes

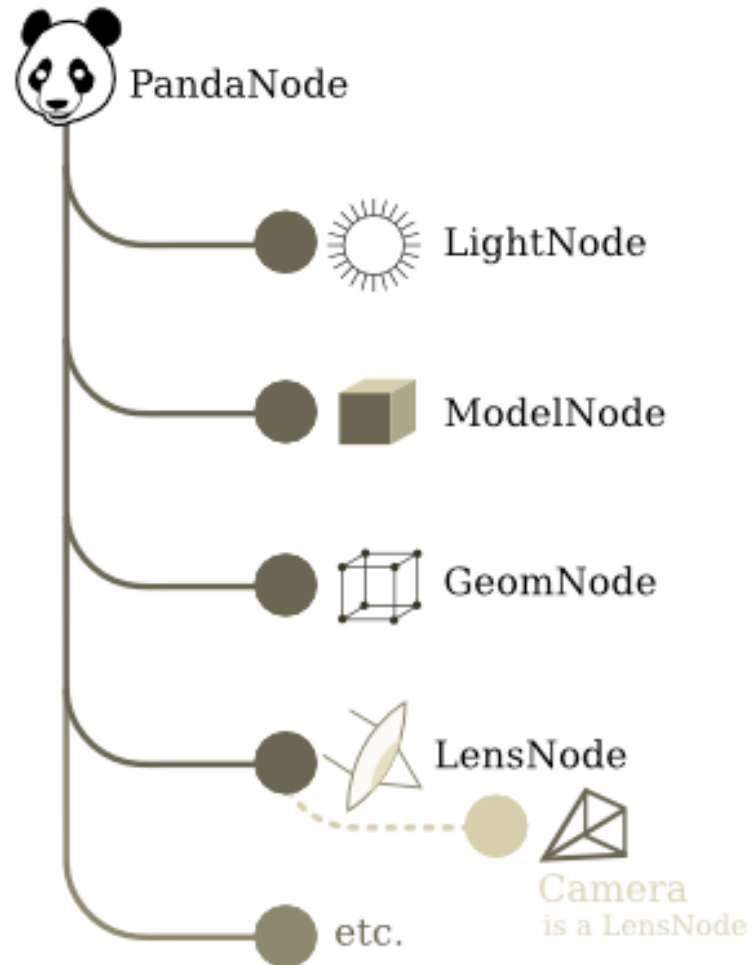


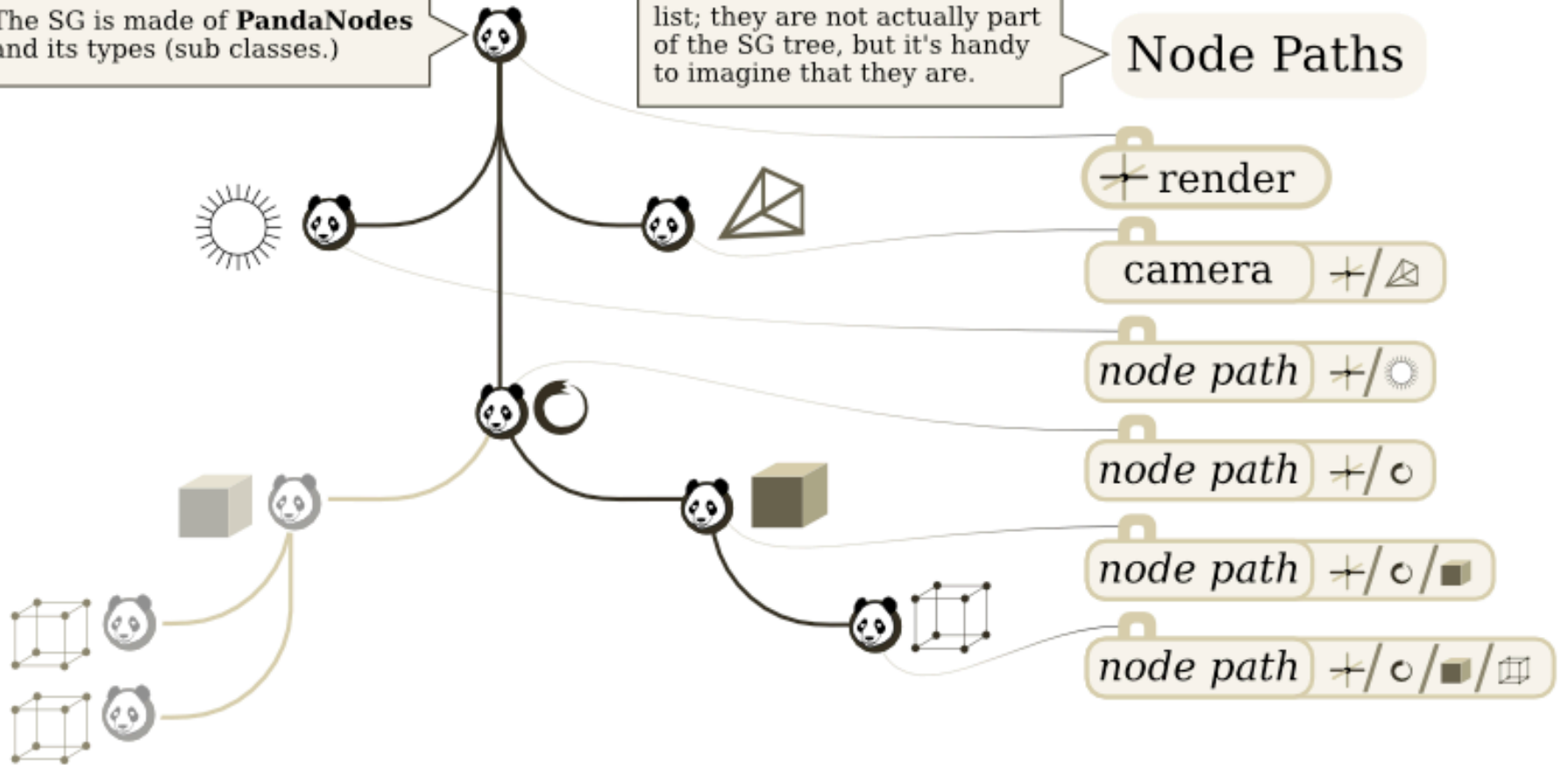
Diagram (c) Donn Ingle 2009 - Creative Commons: NC BY SA

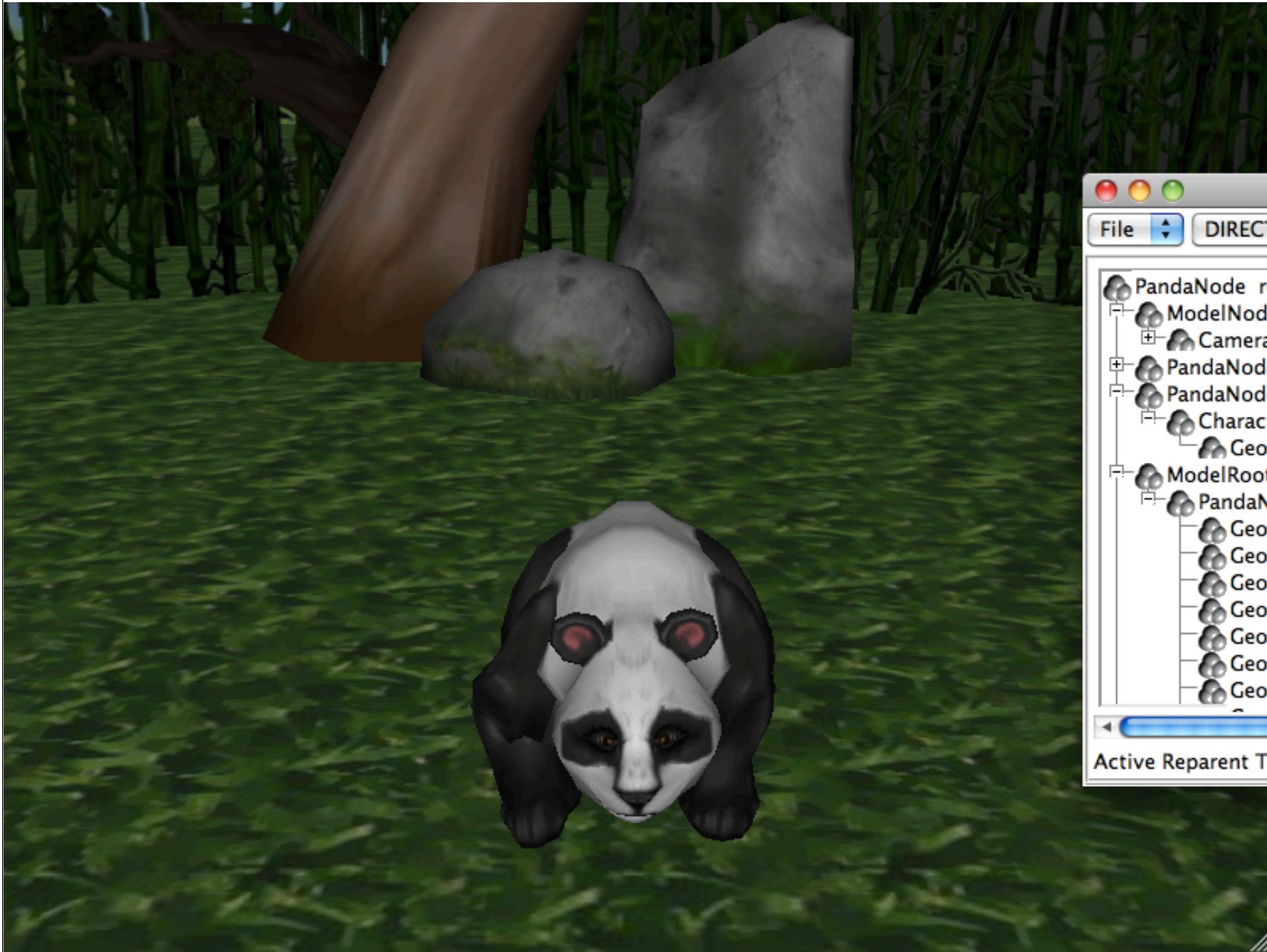
# The Scene Graph (SG)

The SG is made of **PandaNodes** and its types (sub classes.)

NodePaths live in their own list; they are not actually part of the SG tree, but it's handy to imagine that they are.

## Node Paths





tutorial6.py — PyMate

Direct

File DIRECT Help DIR

- PandaNode render
  - ModelNode camera
    - Camera cam
  - PandaNode DIRECT
    - PandaNode panda\_walk\_character
      - Character \_\_Actor\_modelRoot
        - GeomNode
    - ModelRoot environment.egg
      - PandaNode
        - GeomNode TreeTrunk
        - GeomNode Branch2
        - GeomNode Ground
        - GeomNode Branch1
        - GeomNode Branch3
        - GeomNode Rock1
        - GeomNode Rock2

Active Reparent Target:

# Scene Graph

- LightNodes, GeomNodes, ModelNodes, are all subclasses of PandaNode
- Usually, you just need to know about the nodes
- With lights and collision solids, you also need the corresponding NodePath, which is a handle (not a pointer!) to the node. It's a little obnoxious. Sorry

## Collision Traverser

is a task  
that runs  
every frame

Collision  
Traverser

which goes through  
a list of

"from  
objects"

is a task  
that runs  
every frame

which it knows about  
via `addCollider()`

Collision  
Traverser

which goes through  
a list of

is a task  
that runs  
every frame

"from  
objects"

which it knows about  
via `addCollider()`

and compares them  
against all

"into  
objects"

in the scene  
graph

## Collision Traverser

is a task that runs every frame

which goes through a list of

"from objects"

which it knows about via `addCollider()`

and compares them against all

"into objects"

in the scene graph

## Collision Bitmasks

First, it compares the objects'

to determine if these are two objects we care about colliding (by default, we care about all "from" into all "intos")

## Collision Traverser

is a task that runs every frame

which goes through a list of

"from objects"

which it knows about via `addCollider()`

and compares them against all

"into objects"

in the scene graph

## Collision Bitmasks

First, it compares the objects'

to determine if these are two objects we care about colliding (by default, we care about all "from" into all "into")

If the bitmasks have at least one bit in common, test for collision via the objects'

## Collision Solids

which can be added manually or to the model via a MEL script

what do we do if the objects are colliding?

## Collision Traverser

is a task that runs every frame

which goes through a list of

"from objects"

which it knows about via `addCollider()`

and compares them against all

"into objects"

in the scene graph

## Collision Bitmasks

First, it compares the objects'

to determine if these are two objects we care about colliding (by default, we care about all "froms" into all "intos")

If the bitmasks have at least one bit in common, test for collision via the objects'

## Collision Solids

which can be added manually or to the model via a MEL script

what do we do if the objects are colliding?

## Collision Handler

which the traverser knows about via `addCollider()`

## Collision Traverser

is a task that runs every frame

which goes through a list of

"from objects"

which it knows about via `addCollider()`

and compares them against all

"into objects"

in the scene graph

## Collision Bitmasks

First, it compares the objects'

to determine if these are two objects we care about colliding (by default, we care about all "from" into all "intos")

If the bitmasks have at least one bit in common, test for collision via the objects'

## Collision Solids

which can be added manually or to the model via a MEL script

what do we do if the objects are colliding?

## Collision Handler

which the traverser knows about via `addCollider()`

the most common of which is to generate a

## Collision Event

which are "accepted" just like input

## Collision Traverser

is a task that runs every frame

which goes through a list of

"from objects"

which it knows about via `addCollider()`

and compares them against all

"into objects"

in the scene graph

## Collision Bitmasks

First, it compares the objects'

to determine if these are two objects we care about colliding (by default, we care about all "from" into all "intos")

If the bitmasks have at least one bit in common, test for collision via the objects'

## Collision Solids

which can be added manually or to the model via a MEL script

what do we do if the objects are colliding?

## Collision Handler

which the traverser knows about via `addCollider()`

the most common of which is to generate a

## Collision Event

which are "accepted" just like input

each event carries a

## Collision Entry

which is passed to the function specified by the `accept()`

**All Rise**

## Team 1

David Estrella  
Tate Larsen  
Phelan Lemieux  
Randy Sabella  
Ben Shippee

## Team 2

Michael Gruar  
Sean Kim  
Ivy Kwan  
Alan Lummis  
Varun Madiath

## Team 3

Tom Alexander  
Dan Hawkins  
Brett Kaplan  
Andrew Keohane  
Rosa Tung

## Team 4

Kenny Du  
Brian Rana  
David Strohl  
Beth Werbaneth  
Greg White

## Team 5

Eric Collins  
Maria Montenegro  
Colin Neville  
Peter Skinner  
Jared Zondler

## Team 6

Jonathan Brenner  
Alex Devik  
Kyle Johnsen  
Allie Johnston  
Jossued Rivera-Nazario

## Team 7

Adam LeClair  
Angela Mac  
Tyler Moylan  
Anisha Smith  
Casper Tollund

## Team 8

Daniel Cannon  
Ryan Knight  
Thomas Lanciani  
Beth Towns  
Nathan West

## Team 9

Ryan Anderson  
Thomas Anesta  
Joseph Cloutier  
Marshall Hendrick  
Evan Minto

# Assignment #3

- Vehicle combat
  - working headlights
  - collide-able terrain (see Roaming Ralph)
- No humanoid characters!
- Due 11/9