Lecture 11
Announcements
Final 1

- Everyone handed in!
- Don’t forget about previous NC projects
  - They’ll sneak up on you, especially if you have multiple
Playtesting today

• We’ll be playtesting your Final1 handins today
• Afterwards = good time to meet with your mentor TA’s
  – Touch base
  – Make Final2 Rubrics
More on Final Rubrics

• Joint process between you and your mentor TA
  – Your job = present what you think is appropriate for each week
  – TA’s job = check to ensure it keeps you on good pace

• Ideal rubrics:
  – Final1 – engine features partially done, some possibly demo-able
  – Final2 – engine features completely done, all demo-able, limited gameplay implemented (< MVP)
LECTURE 11
Procedural Content
What is procedural content?

• Any content that is generated algorithmically
  – Terrain/biomes
  – Cities
  – Objects/Artwork
  – Characters
  – Quests
  – Puzzles
  – Dialog
Why procedurally generate?

• Cons:
  – More effort to get initial content
  – Less predictable, more susceptible to tricky bugs

• Pros:
  – Replayable
  – Less effort to make a ton of content
  – (Can be) easy to tweak towards desired effects
  – You can more accurately playtest your own content
    • Normally you’re biased by prior knowledge (ex. levels/puzzles)
  – It’s fun! You can make art with code!
Procedures are algorithms

• Often based around a noise function
  – Perlin, fractals, sine, rand, etc.
• Noise can directly translate to visible values
  – Height, temperature, color, character stat
• It can influence decisions from a set of options
  – Block type, enemy type, path generation, behavior
Procedures are algorithms

• Can also be deterministically created based on fixed initial content
  – Many older games used this because of memory constraints

• Can track player decisions and use as input to determine more content
  – Available quests (ex. choosing a faction in Skyrim)
  – Enemies suitable or challenging for current stats, weapon types, playstyle
Online vs. Offline

- **Online** – generated during gameplay
  - Essentially infinite, continuous generation
  - Generate new content for each play through
  - Can adapt to playstyle

- **Offline** – generated during development
  - Can be tweaked/extended by designers
  - Use for design inspiration
  - Algorithm doesn’t need to be fast
  - Doesn’t have to guarantee validity
Validity

- Make sure your content works
- It should look natural in your game’s aesthetic
- No outlier or ‘bad’ maps, even if playable
QUESTIONS?
Procedural Content

LEVELS
2D Perlin Noise

• Basic height map
  – Combine frequencies for better variation

• Biome partitioning

• Can also combine with other 2D noise functions
3D Perlin Noise

- 3D terrain shapes
  - Could combine with 2D height map
- Subtraction
  - From height map
  - Mazes
  - Perlin Worms (caves with distance constraints)
Dungeons in Grids

- Simple connected grid
- Binary Space Partitions
  - Divide area into spaces, put a room in some, and connect
- Combine grids and paths
Left 4 Dead 2 – The Director

• The terrain layout in L4D2 is set, but other content is generated procedurally:
  – Mini-bosses
  – New weapons
  – Health/Ammo

• The Director assesses the player’s “situation”:
  – Location
  – Health
  – Skill (accuracy, time
  – Emotion (ex. mouse accelerations,
    reaction speed, key press rate)

• Tries to engineer the best experience
Geometric vs. Voxel

- Lots of these algorithms are much easier to implement and tune given discrete coordinates
- Making freeform structures or levels requires more constraints to line things up
- Also want some kind of smoothing noise for blending
- Despite challenges, more room for creativity!
Levels

QUESTIONS?
OTHER PROCEDURAL CONTENT
Weapon Generation

• Randomize stats
  – Ex. Guns: damage, recoil, particle speed, clip size, fire rate, etc.

• Pick type from preset list
  – Randomize specific attributes of chosen type

• Chose appearance from random combination of traits
  – Randomly built names
Weapon Generation

• Generate bullet design

• Galactic Arms Race:
  – List of mixable attributes defining shot type
  – Adjustable functions for trajectory
  – Weapon drops adjust for your weapon preferences
 Enemy Generation

- Starbound generates random creatures on each planet
  - Colors fit creature’s biome
  - Parts are mixed randomly from many existing assets
  - Attacks are also chosen semi-randomly
Stat Generation

- Pokémon are born with hidden IVs (‘individual values’) that determine stat growth rate
- ‘Effort values’ affect stats based on the enemy defeated
- Pokémon also have ‘natures’ to influence stats
- Also shinies
Procedural AI

• List of all possible behaviors for an entity
  – Each has its own specific customizations
  – These can be random or influenced by other entity stats, like ‘personality’ or ‘aggression’

• Procedurally choose its graph of behaviors
  – Semi-randomly weighted probabilities and/or triggers to switch states
  – Procedural learning – every time the entity takes damage in a behavior, it becomes less likely to perform it
Procedural Generation vs. AI

• Lots of ‘procedural content’ strategies overlap with game AI solutions
• Try adapting other AI structures for other kinds of procedural content!
• Add procedural/semi-random components to your AI too!
QUESTIONS?

Other Procedural Content
Creating your own procedure

• Start with concept art or descriptions of end goals – what do you want to make?
Determine your requirements

• What steps do you need to get there?
  – What shapes or subdivisions of shapes will help?
  – What variations will change one random desired image/chunk/product to another?
  – Break down each step until you know the specific problem you are solving, and find a solution
Example: Complex enemy

- Enemy goals:
  - 1-3 creatures riding around together
    - So each creature needs a rider capacity, and visual mount points
  - Similar enemies should be paired together
    - So each creature should have one or more types to match with
    - Define similar/complimentary color palette for all creatures
Example: Complex enemy

• Enemy goals:
  – Bodies should be visually distinct
    • Mix up required body parts
    • Create various templates for where to attach body parts (ex. two heads ->)
    • Create lots of small parts for each type
  – Attacks should fit theme
    • Give attacks types like enemies
    • Find best match between all paired enemy types and attack types
Example: Small route

• Terrain goals:
  – Gentle maze with somewhat distinct ‘rooms’
    • Use a known algorithm like Perlin subtraction or BSP with fuzzy noise around the edges
  – Run a subtle path from start to finish
    • Use A* to find best path
    • Add noise to the path
    • Only draw the path when it overlaps with another Perlin function
Example: Small route

• Terrain goals:
  – Make each room ‘interesting’
    • Use other noise functions to add alternate tiles like flowers or tall grass
    • If tile is far enough from A* path it can choose an obstructing tile
    • Occasionally add some preconstructed ‘features’ like ponds
      – Which can have procedural attributes as well! Yay!
Good resources

- http://pcg.wikidot.com/
- http://pcgbook.com/
LECTURE 11
Maya and 3D Software
So you need assets...

- How will you make them?
- Characters, environment assets, etc.
- Your engine allows for loading OBJs already, so why not make your own?
But how do I make them?

- There are a lot of 3D modeling programs out there to make 3D Assets
  - Maya, Blender, Zbrush
  - Different programs have different benefits
- We will focus on how to do basic modeling and shading of assets in Maya
  - Free for students!!
  - In general, good for all steps of the asset production process
- Download here: http://www.autodesk.com/education/free-software/maya
The Maya GUI

**Primitive editor and channel editor** can be brought up from Window -> Outliner.

*Important*

**Navigation tools**

**Dropdown menu editor**
Shortcuts

- Left click to select objects
- Right click and hold to bring up marking menu
- Left click + alt to look around
- Middle mouse click + alt to pan
- W – translate
- E – rotate
- R – scale
- F to focus on selected object
- 1 – view geometry unsmoothed, 3 – view geometry smoothed, 4 – wireframe views, 5 – shaded view, 6 – shaded display with texture maps
Creating Assets

MODELING
Basic Modeling Operations

• Moving particular edges and vertices
• Extruding
• Combining meshes
• [demo]
Modeling tips

• Geometry likes quads
  – These will deform in less unexpected ways when you animate a character
  – Smoothing cubes vs. using spheres

• Soft select

• Orthographic views and sketches

• Use a mouse for middle click!

• SAVE FREQUENTLY for the love of god
Creating Assets

SHADING
So now I have a gray cube

- How do I make it a colorful cube?
- UV Mapping
  - Planar mapping – X, Y, Z
  - UV Editor
- Assigning new materials
  - Lambert, Blinn, etc.
  - Hypershade
Some examples

• Let’s look at some of Miranda’s animation projects lol
Character Modeling

• Usually modeled in the standard “T-pose”
• In order to animate this static object, we need to rig it and skin it
• Rigging is essentially constructing a working skeleton
• Skinning is making sure the character mesh moves with the skeleton
Example: Lightbulb Girl

• More of Miranda’s projects aaaa
Rigging and Skinning Tips

• This takes a hell of a lot of time, so make only a very simple skeleton if you are planning on animating a character

• Tutorial: https://www.youtube.com/watch?v=AkJk7d30ks

• Save all the time. Do it.
Resources

• Maya tutorials: https://knowledge.autodesk.com/support/maya/learn-explore/caas/simplecontent/content/maya-tutorials.html
• Doing the first few should get you well enough acquainted with the environment/simple modeling
• Do a lot of googling!! Oftentimes entering the question into the search bar will get you the answer
• There are so many youtube tutorials out there. They are incredibly useful
• Once again SAVE ALL THE TIME. This program can freeze and crash and make you cry and you can lose all your work like Miranda has done a thousand times over
Creating Assets

QUESTIONS?
LECTURE 11
Skeletal Animation
Skeletal Animation

• Consists of joints and/or bones to represent parts of the object
• By itself, provides no shape deformation, only translation & rotation
Skeletal Animation

• This works great in Minecraft, where everything is a rectangular prism anyways

• It can work with arbitrary mesh pieces too
  — Just watch out for meshes intersecting
Example Joint System

• Create a root joint, then hierarchically add joints with fixed offsets
  – the space between root and child joints is the ‘bone’ area
  – assign mesh(s) to render at each joint or bone
  – keep track of the rotation(s) active at each joint
    • You should only need three (x, y, z axes of rotation)
Hierarchical Joint Systems

- In more generality, skeleton represented as a hierarchical graph
  - Each joint is a node, may or may not have children
  - Store transformation at each joint relative to parent joint
    - Global transformation of child $=$ Global transformation of parent $*$
      Local transformation of child relative to parent
    - Like CS123 sceneview
- OpenGL can perform the rendering rotations for you easily
- It’s not a ton of work to set up — expect to spend a good amount of time on content creation to make animations that look nice
Animating joints

- Think of some tools that you can use to specify an animation
  - Keyframes (or timers)
    - At time $t_0$, joint $j_3$ rotation $x$ is $x_0$
    - At time $t_1$, joint $j_3$ rotation $x$ is $x_1$ (w/ linear interpolation)
    - At time $t_2$, joint $j_3$ rotation $x$ is $x_2$ (w/ quadratic interp)
  - Programs like Maya have these features and many more – look for inspiration to make something complex
Animating joints

— Functions
  • Set joint’s rotation $y$ to $A*\sin(B*time)$ (rotate in a circle)
  • Might be useful for a looping walk animation

— Combination of tools
  • Ex. Joints animate with a function while character is moving, then interpolate back to resting position

— Abstraction: build small animation pieces (ex. ‘swingArms’, ‘bounceTail’) and combine them for more complex behaviors

— Inverse Kinematics
  • Just kidding
Skeletal Animation

QUESTIONS?
Rigged Animation

- Consists of a skin (mesh) binding on top of a skeletal/joint system
- Useful for any animation that deforms a mesh
Rigged Animation

• Given a set of vertices representing your mesh, and a set of joints or bones:
  – Determine which vertices correspond to which bones
  – Determine how much weight each bone has on its vertices
  – Calculate how the changing bone positions affect the vertices they influence

• The first two are probably content creation, the last is your engine feature
Rigged Animation

• One option is manipulating vertices with bones:
  – Each vertex has a vector offset from each bone it is attached to
  – As the bone translates around, maintain that same offset for each vertex
  – As the bone rotates, apply that rotation to each vertex with its offset
  – This may be slow for a detailed model
    • Look into using matrices to speed up the computations
    • Use a program like Maya to optimize your models
Rigging (Skinning) a Mesh

- Skinning is the technical process of assigning bone/joint ‘weights’ to each vertex
- Each vertex can have up to a set number (usually around 3-4) bones that affect it
- Traditionally, this would be done by hand in a program like Maya
  - An algorithm assigns initial weights with some heuristic
  - The animator then tweaks these weights to look right
Rigging (Skinning) a Mesh

• You should implement your own algorithm for assigning bones, then try tweaking the weights
  – Start with a basic distance heuristic (assign vertices to the bone(s) closest to them)
  – Add conditions, such as: multiple bones affecting a vertex should be hierarchically connected

These vertices might choose the wrong leg bone as the 3rd closest bone
Rigging (Skinning) a Mesh

- Transform vertices by linearly interpolating transformation at associated joints
  - Let v be a vertex with joint weights w1 and w2 for joints 1 and 2
  - Let v1’ be the position of vertex v relative to coordinate system of joint 1 in the initial mesh configuration
  - Let v2’ “ of joint 2 “
  - Let T1 be the transformation of joint 1, T2 be the transformation of joint 2
  - newVertexPosition = w1 * T1 * v1’ + w2 * T2 * v2’
Final1 playtesting

Yay!