# Assignment 4 Overview


MMD File Format

# Collada File Format

Collada is open standard for 3D meshes and animations

- More flexible and accessible than MMD

Please feel free to create your own models and rigs if you are artistically inclined (or just really love Vocaloids)

# Parsing Joints and Weights

Collada data loaded in via AnimationFileLoader

Scene contains a number of useful classes:

- `Attribute` (Mesh attribute information)
- `MeshGeometry` (Face and vertex information)
- `Bone` (Bone information)
- `Mesh` (Mesh and bone information)

# Visualizing Bones

- Bones rendered via `skeletonRenderPass`
- Must add shader for highlighted bones (i.e. render a cylinder cage around the bone)
- Cylinders represented by two types of lattices:
  1. Line segments following the length of the bone
  2. “Circles” (really just n-gons) at the bone’s end and middle positions

# RenderPass

- Class for handling all the OpenGL setup/ boiler plate
- Provides abstracted container for working with VAOs, VBOs, programs, attributes, etc
- Not essential to understand but may be helpful

# Bone Picking

How to select an object in 3D (world) space based on the mouse position in 2D (screen) space?

# Ray Casting

Generate a ray from screen space coordinates to world space (ScreenToWorld)

How to do this?

# Camera Matrix

- Compute the NDC (normalized device coordinates) from the screen position
- Determine coordinates in world space based on camera position and orientation

# GLM::Unproject

Same idea as reversing the camera matrix (i.e. using the inverse), but also handles NDC conversion out of projection space

Must create a ray based on this unprojected position in world space

- Subtract eye position
- Unproject two positions along z to find ray

# Cylinder Intersection

Intersection must be within cylinder radius and height

Intersect with cylinder in one of three ways:

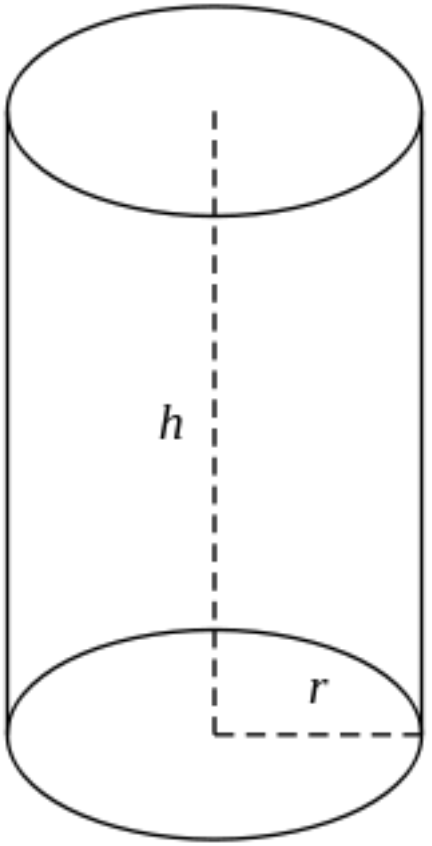
1. Intersect based on cylinder's position in world coordinates (inefficient)
2. Intersect based on cylinder's position in bone's local coordinates (transform ray into this coordinate system)
3. Intersect based on 2D projection of cylinder (check if hit within radius then compute intersect points to check height)

# Parametric Cylinder Intersection

A cylinder can be considered a sequence of disks

Equation for infinite unit cylinder along Z-axis:

$$x^2 + y^2 - 1 = 0$$




Substitute parametric ray equation:

$$(O_x + d_x t)^2 + (O_y + d_y t)^2 - 1 = 0$$

# Parametric Cylinder Intersection

$$(O_x + d_x t)^2 + (O_y + d_y t)^2 - 1 = 0$$

- Expand and solve for t as a quadratic equation
- Check for extent of finite cylinder based on height of bone
- Assumes ray intersect happening in cylinder's local coordinate system

# Object Picking Using Pixel Color

Kind of a hack but a classic!

- Create a buffer (not seen by viewer) that renders all objects as different colors
- Map user's screen selection to this buffer and check pixel's color

http://www.lighthouse3d.com/tutorials/opengl-selection-tutorial/

# Bone Manipulation

GUI class handles keyboard and mouse callbacks

Mostly similar to what you did in Menger…

…except you'll also need to update the bone in the deformed coordinate system (will want some additional handling in `Mesh`)

# Linear Blend Skinning

Final step is to connect the vertices (stored in `MeshGeometry`) to the deformations being applied to the bones (stored in `Bone`)

Performed on the GPU in the vertex shader (`sceneVSText`)

# MeshGeometry

Skin weight information stored in `MeshGeometry`

- `skinIndex` (index of influencing bone)
- `skinWeight` (weight of influencing bone)
- `v0-v3` (position of mesh vertex in influencing bone's coordinate system)

# LBS Vertex Shader

All attributes you need are already being passed into `sceneVSText`

- Apply bone weight for all influencing bones (capped at 4) to adjust vertex's position in **world space**
  - Must use bone's TR information and and local position information to get the vertex's updated position
- Note: `vertexPosition` is a placeholder
  - Won't need it once LBS is applied

# Additional Caveats

All join rotations are handled as quaternions!

- Must work in quaternions even if you are not apply DQBS
- Look over ThreeJS Quaternion library to see available functionality/built-in Euler conversions

# Design Document

Start working on a design document that addresses where in the code base and how you will tackle A4:

- Summarize the skinning pipeline and identify where your changes will go
- Describe additional classes you might create
- Describe changes to existing classes that will be helpful
- List any shaders/shader functionality you need to create
- Make sure to clarify the math necessary (i.e. transforms and coordinate systems) for each of these features

We will peer review each other's documents on Friday, so this exercise is both for your benefit and the benefit of your classmates!