UCLA

CS174A : Introduction to Computer Graphics

Royce 190
TT 4-6pm

Scott Friedman, Ph.D
UCLA Institute for Digital Research and Education

Introduction

- Scott Friedman
  - MS, Ph.D in Computer Science
  - B.Arch, M.Arch in Architecture
    - Founding member of UCLA Urban Simulation Team
      - » http://www.ust.ucla.edu
  - Thesis on Real-time parallel rendering
  - Chief of Technology for UCLA Research Computing
    - High Performance Research Computing, Networking and Storage

Introduction

- Class Tas
  - both are graphics Ph.D students
- Xiaolong Jiang
  - PhD Student
- Franklin (Jingyi) Fang
  - PhD Student

Introduction

- Contact
  - You should feel free contacting us by email.
  - Use the course forum (CCLE or Piazza?)
    - Good for common questions/answers
  - If you need to see me –you must- set up an appointment.
    - Office is 3344 Math Science Building
    - Office Phone is 310-825-8607
    - Email is best. friedman@ucla.edu
  - Text
    - Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL, **7th Edition**, Angel & Shreiner

Course Administration

- Four* Assignments (Programming)
  - * One “get your feet wet” exercise
  - Three progressive that build on each other
  - Extra credit *always* offered
- Mid-Term Exam (optional)
- Term Project (Programming)
  - This will be the meat of your experience!
- Final Exam (optional)

Course Administration

- Grading
  - We will use a point system
  - 1000 points would be the max for assigned work
  - Extra credit available on programming work
  - We will grade on a straight percentage
    - 900 = A, 800 = B, 700 = C, and so on.
  - Specific tasks assigned points.

Course Administration

- Programming Assignments
  - (100 points ea., 300 total)
    - First one will get your development environment up and running (**required**, but no points).
    - Basic example program based on Chapter 2
    - Extra Credit options will be available for all four assignments.
    - Exact details of what you have to do will be outlined in the assignments.

Course Administration

- Programming Assignments
  - Final three assignments will exercise your understanding of the course elements to prepare you for the term project.
  - They will build on each other so keeping up with the assignments in important.
  - More extra credit will be available here.

Course Administration

- Mid-Term Exam (100 points)
  - Optional?
  - Yes, if you feel you are earning enough extra credit points elsewhere – sleep in!
  - The test will be open book
  - Closed laptop/phone/network, everything else
  - Concept based: multi-choice, short answer
  - If you are completing the assignments you should be just fine.

Course Administration

- Final Exam (200 points)
  - Same arrangement as the Mid-Term Exam
  - If you are attending the lectures
  - Doing the assignments and project
  - Asking questions
  - Using your brain
  - You should be just fine

Course Administration

- Term Project (400 points)
  - You ***will*** write a ***non-trivial*** graphics program.
  - It ***will*** include some type of user interaction.
  - You ***will demonstrate*** your work ***live*** in front of class.
  - You ***will*** work in a team.
    - Teams are three to five people.
  - All team members will receive the same points
  - Requirements ***will*** go up proportionally to the number of people in your team!

Course Administration

- Term Project (400 points)
  - Start thinking about this **now**
    - Team? Ideas?
  - Be careful not to get yourself into a bind.
  - Extra credit will be available (more later)

  - You ***will*** all vote on each others projects during the demo sessions during the final week of class!
  - It will be fun…right? :)

Course Administration

- Term Project (400 points)
  - A little additional motivation for you all
  - The top five (based on voting by your peers) will get bonus points in addition to your grade and extra credit.
    - First place = 200 points
    - Second place = 150 points
    - Third place = 100 points
    - Fourth place = 75 points
    - Fifth place = 50 points

Course Administration

- Something to Remember
  - If you do the Assignments
  - Take the Exams
  - Complete the Term Project
  - …and do not bother with extra credit
  - It is still possible to get an A
  - The extra stuff is for those of you who
    - Want to challenge yourself (or show off)
    - Avoid the exams
    - Have some extra fun

Course Administration

- Expectations
  - You pay attention and show up
  - You do your work with integrity
  - You stay off your phone/texting/chat/twitter/…
  - You are proficient programmer
    - You will become one by the end of this quarter whether you like it or not! ☺
  - You can set up a development environment
  - You have some idea of how to debug
    - The TAs ***will not*** debug your work for you
  - You really need your own computer
    - You should be OK using the SEAS labs

Course Administration

- Development Environment
  - Again, you really should have your own stuff
    - You do don't you? Less fun if you do not…
  - I am being unreasonable about this?
  - Linux, OSX, Windows all ok – I do not care
  - We will be using WebGL in this class
    - Should simplify things…
    - You can do your final projects on whatever, using whatever you like. Our ability to 'help' you will be directly proportional to our own experience.

Course Administration

- Purpose of Class
  - Learn basic computer graphics and interaction
  - Use ***all*** of your CS skills
  - Help each other
  - HAVE FUN!
- More detail in Syllabus
- Questions?

A little motivation…

- Computer Graphics? How boring!

A little motivation…

- Computer Graphics? Not! It is

A little motivation…

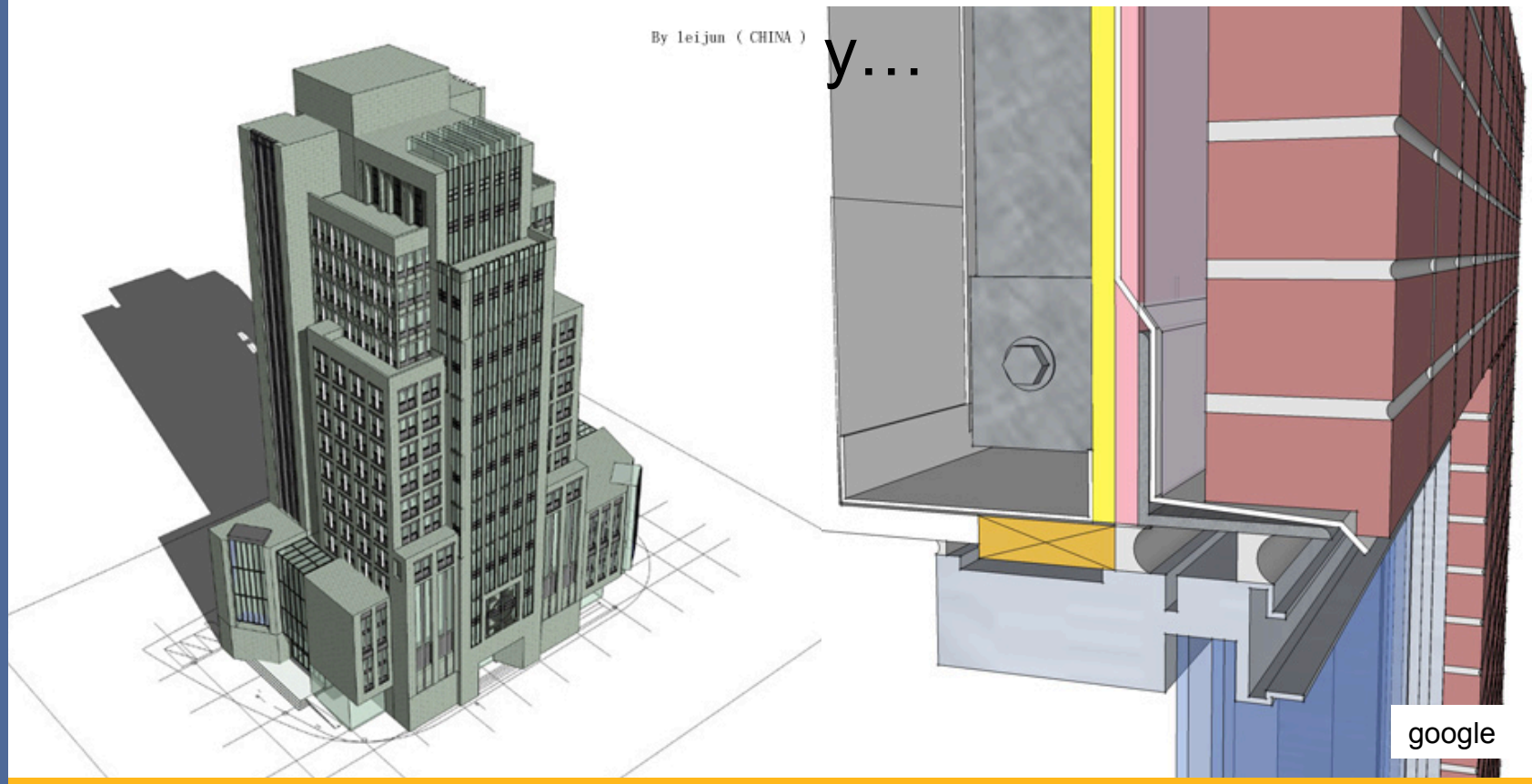
- Movies!

A little motivation…

- Games! …a larger business annually than the movies
  - It is everywhere today…

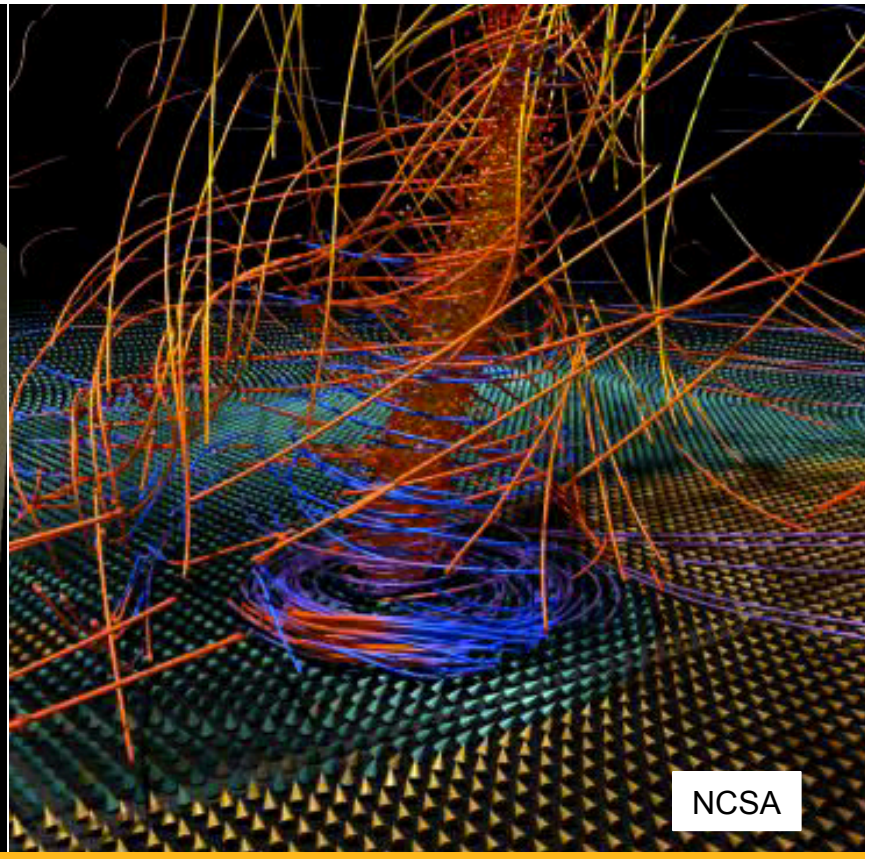
A little motivation…

- CAD!



A little motivation…

- Scientific Visualization!



A little motivation…

- Simulation and Training!

Lockheed F35 Trainer

A little local history…

- UCLA not typically known for graphics
- Or is it?
- UCLA Architecture and Art
  - Home of 'Processing'
  - Robert Abel (one of the first customers of SGI)
  - CAD Pioneers
  - Early developers of VR technology

A little local history…

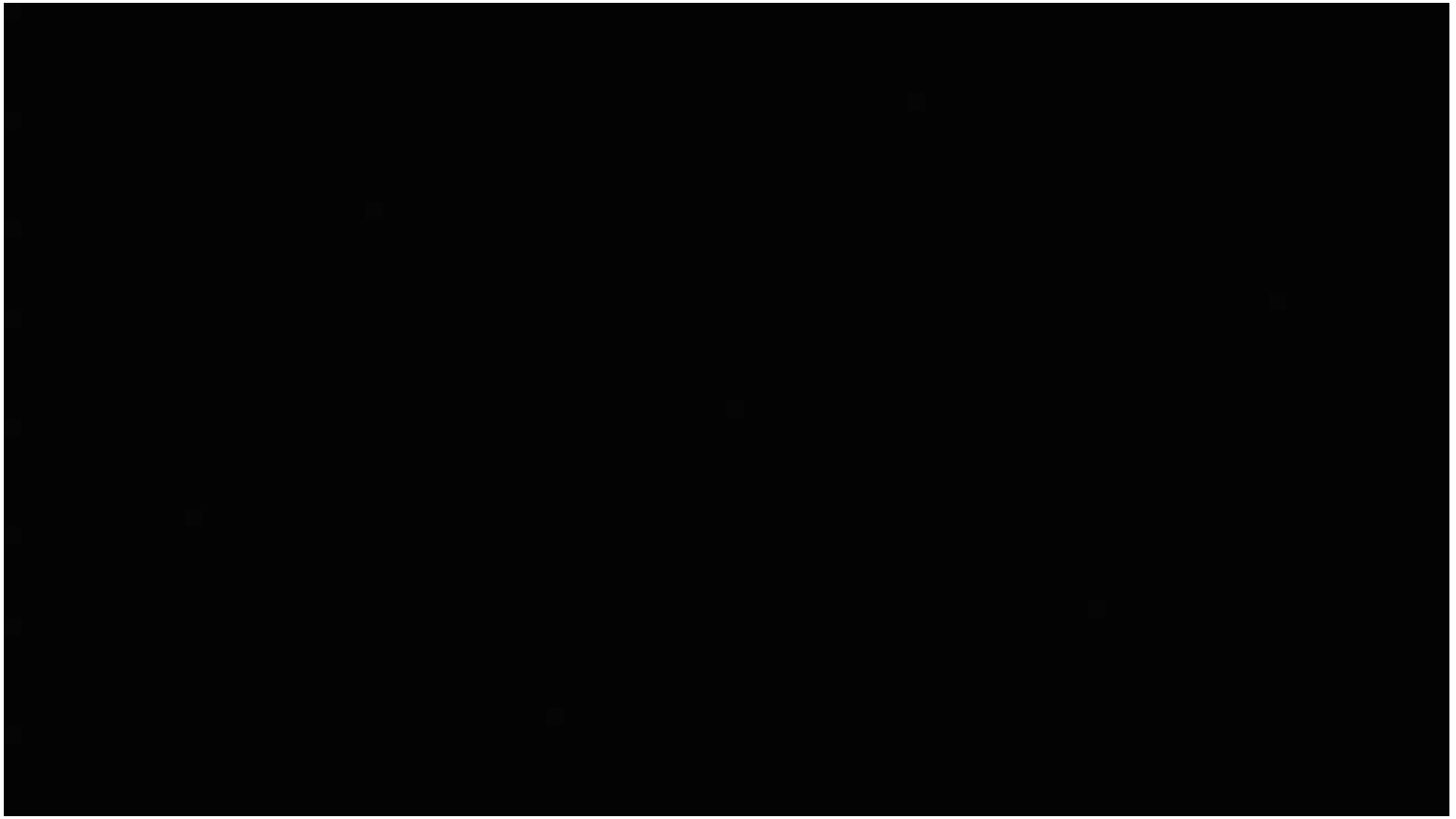
- How Early?
- NASA Apollo Lunar Docking Simulator
  - Built by General Electric in the late 60s
  - Render 40 polygons per second in real-time!!!
  - In color no less!
  - Later evolved into the Compu-Scene IV
    - First device to support texture mapping in hardware
    - Most successful fight simulator ever

A little local history…

A little local history…

- City-Scape led directly to the founding of the Urban Simulation Team at UCLA
- In the early days (20 years ago) we used big, expensive SGI computers.
- There have been more than 40 people who have been part of the Team (lab)
- The lab, unfortunately, is no more ☹

A little local history…



Let's get started…

- Top-Down Approach?
  - Means we will start at the API level and explain the internals and theory, as needed, as we go.
  - Means you will have something on the screen very quickly – in the next week.

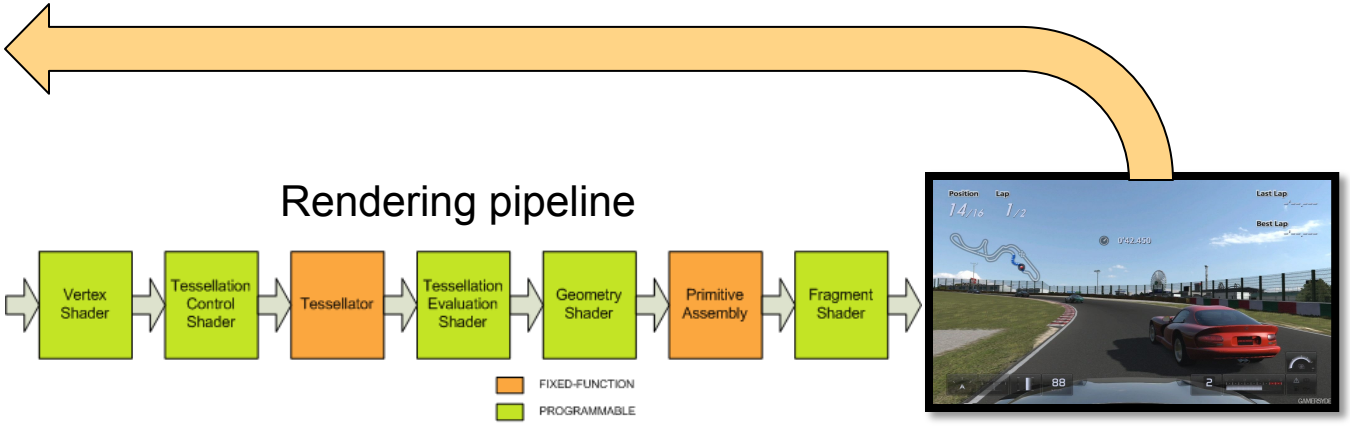
Let's get started…

- Two concepts are integral to the class

- Interaction (or event) Loop
- Imaging Pipeline
  - We will be using shaders in this class
- Understanding of the camera model

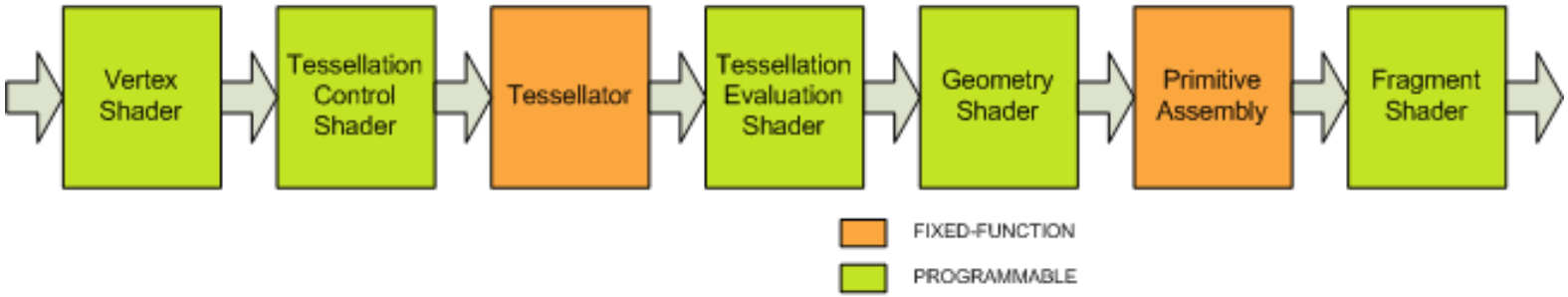
Interaction

- Interaction (or event) Loop
  - Collect User Input
    - From a mouse, joystick, finger, etc.
    - Send data to be displayed down the “rendering pipeline”
    - Display the result
    - Repeat



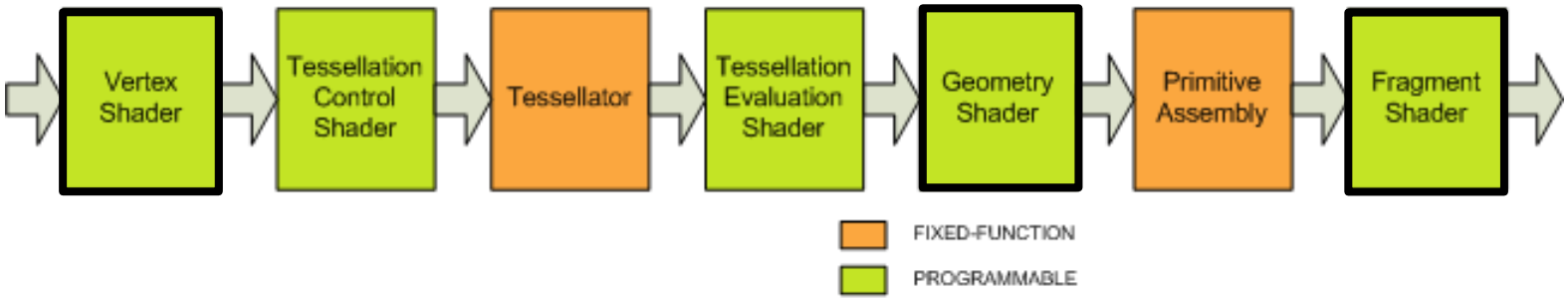
Rendering Pipeline

- Rendering Pipeline (OpenGL)
  - State Machine – old style fixed function
    - Colors, Textures, Lighting, think 'knobs' you turn
  - Shaders, think of little programs instead of simple 'knobs'
  - Geometry data passed in…
    - Projected, Rasterized, Colored (Visibility) and Displayed



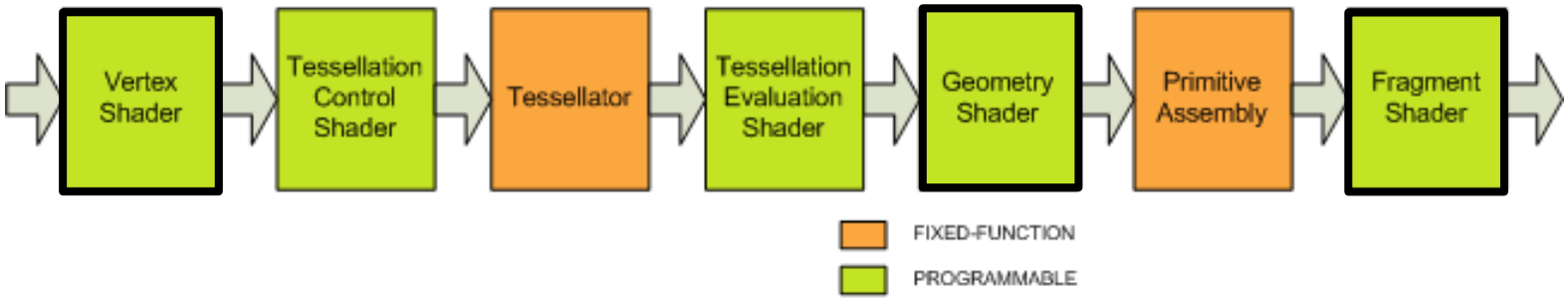
Rendering Pipeline

- Rendering Pipeline (OpenGL)
  - More complex than previous versions of OpenGL
  - This is the future (now) as all graphics hardware is programmable. (even on mobile devices)
  - Green boxes are programmable
    - We will focus on the highlighted boxes in this class
      - » Vertex, Geometry and Fragment



Rendering Pipeline

- Rendering Pipeline (OpenGL)
  - We will be writing little programs (shaders) that are executed on the graphics hardware to perform these different functions.
  - The thing to understand now is
    - The pipeline is hardware we access through the OpenGL API that takes geometry as input and produces images as output.



Camera Model

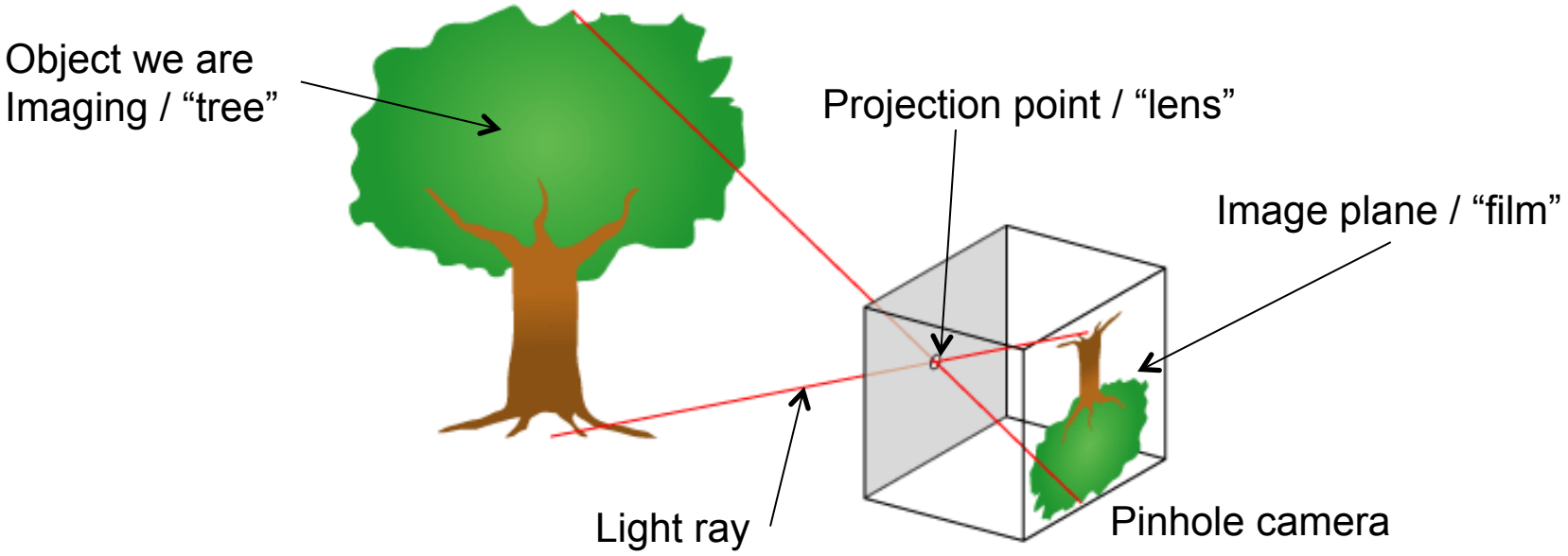
- The most important concept to “get” is how we are forming an image on the display.
  - If you think of a camera, think of how an image is “captured” on the film (a plane) as a metaphor.
  - In the computer we do not have light rays so we will approximate the same effect using math (geometry and linear algebra)
  - This will allow us to do quite a few things you could never do with a real camera!

Camera Model

- Using our camera metaphor what pieces do we need to form an image?
  - We need an object to form an image of
  - We need some kind of “lens”
  - We need someplace on which to form the resulting image, some “film”
- In a real camera the light rays “project” through our lens to form an image on the film.

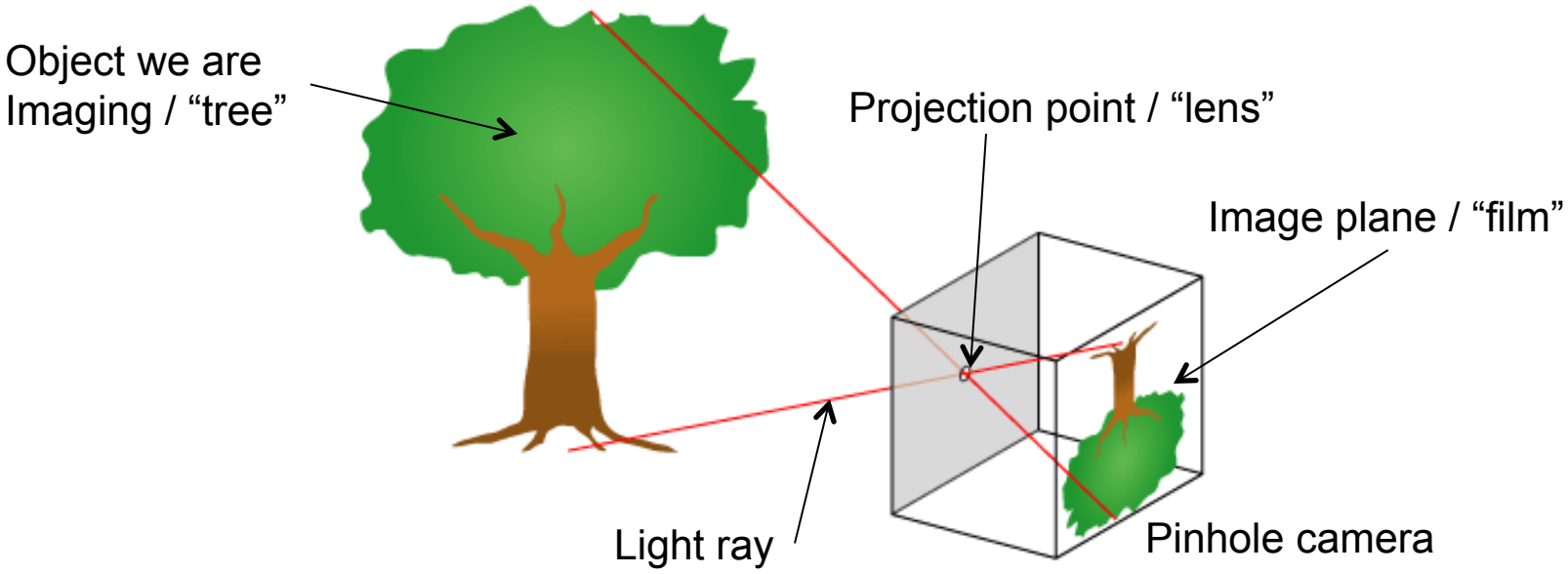
Camera Model

- A pinhole camera has a very simple "lens"
  - A tiny hole to let light rays through
  - Lets call this the "projection point"



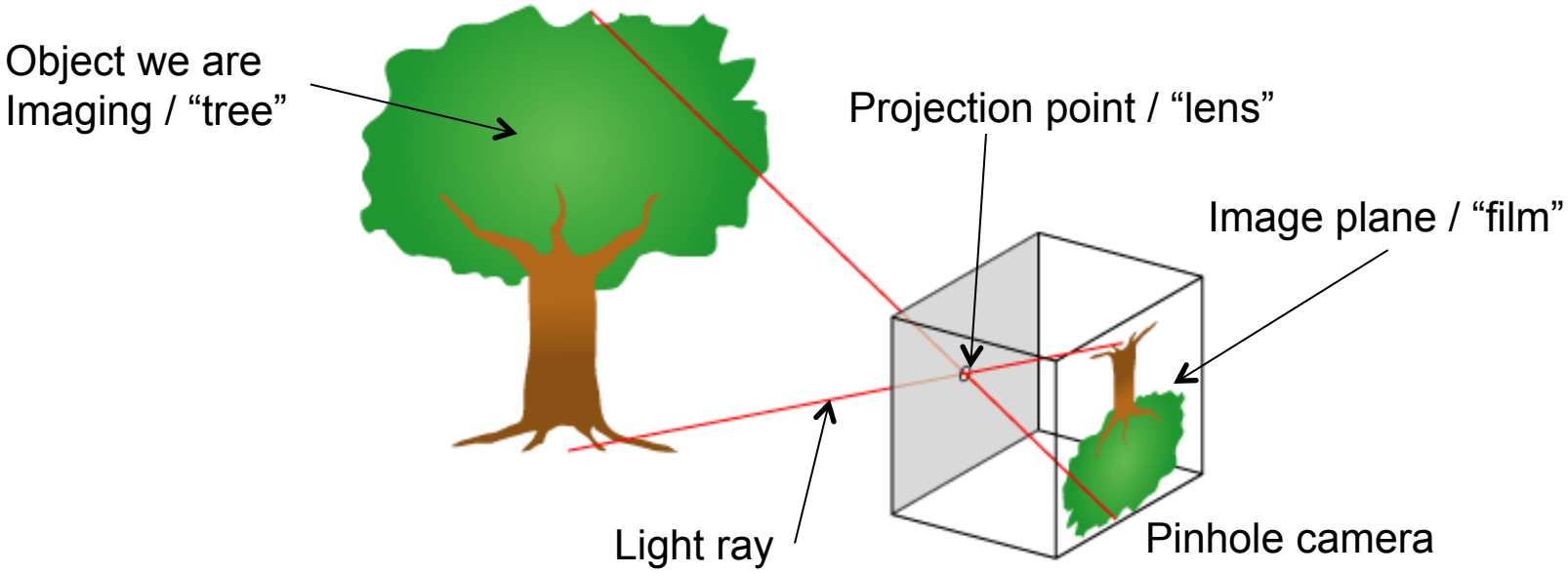
Camera Model

- Let's turn this around
  - Let every point on the "film" capture what it can "see" through the projection point (lens)



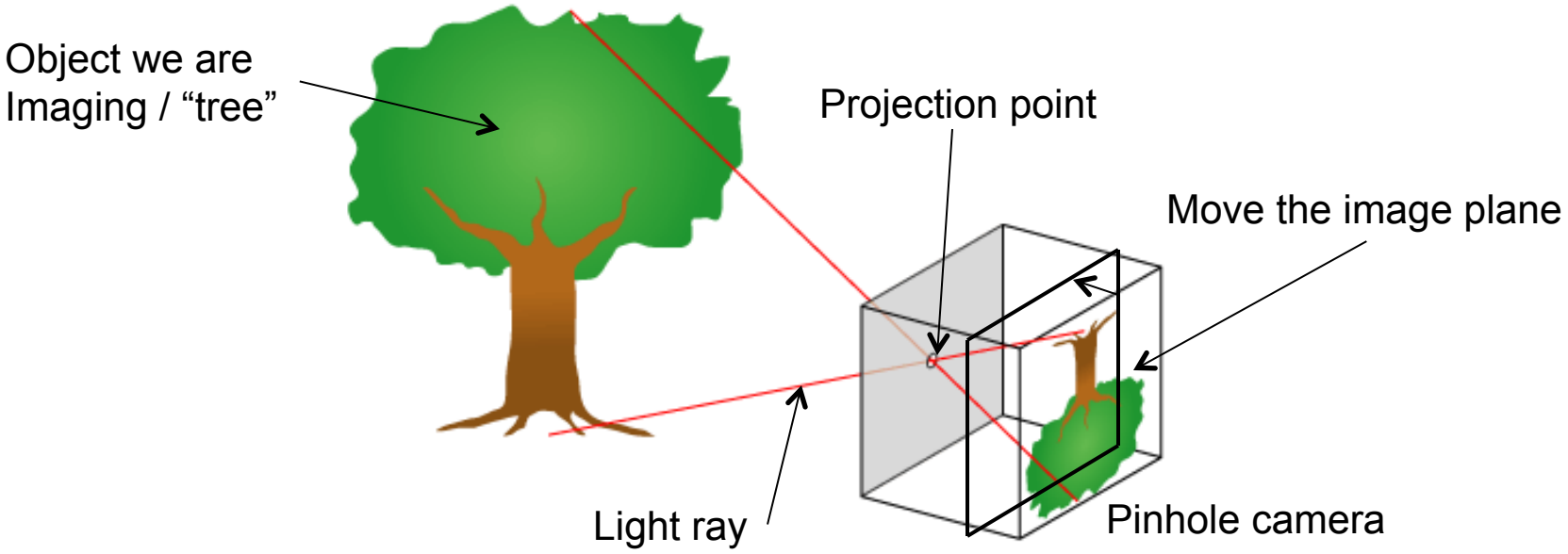
Camera Model

- Can you imagine what would happen to the image of the object if we moved the image plane (film) relative to the projection point?



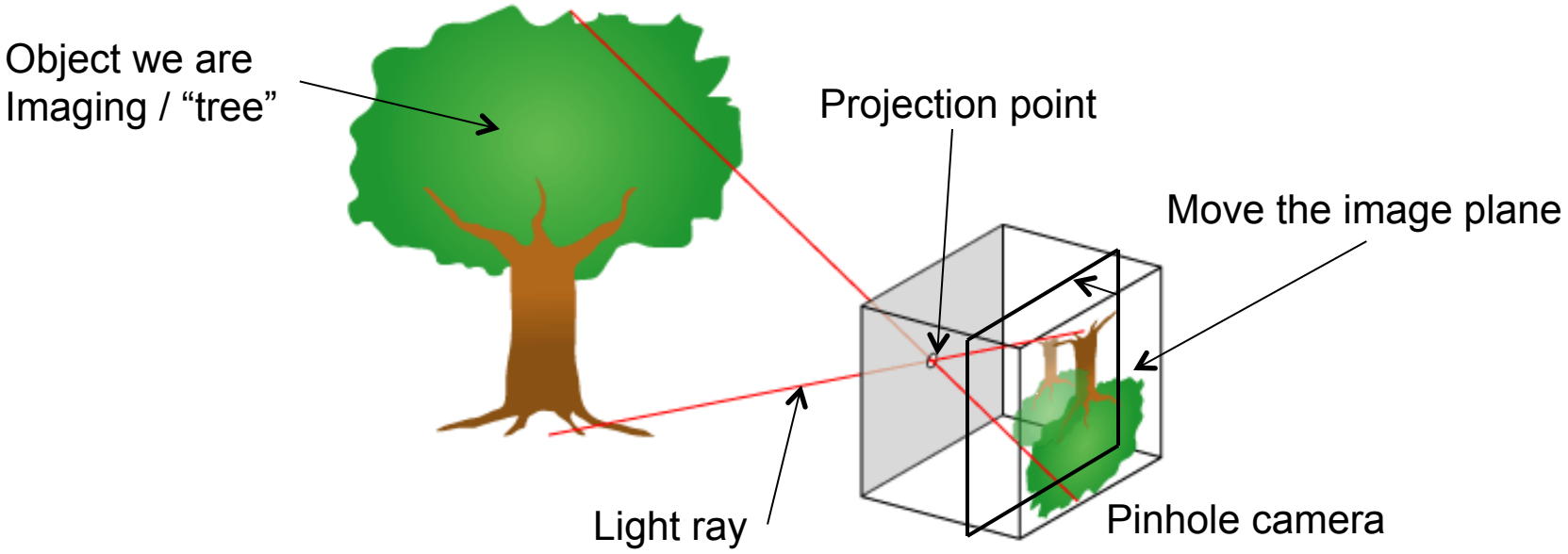
Camera Model

- As we move the image plane *closer* to the projection point, what would happen to the *object* we are imaging?



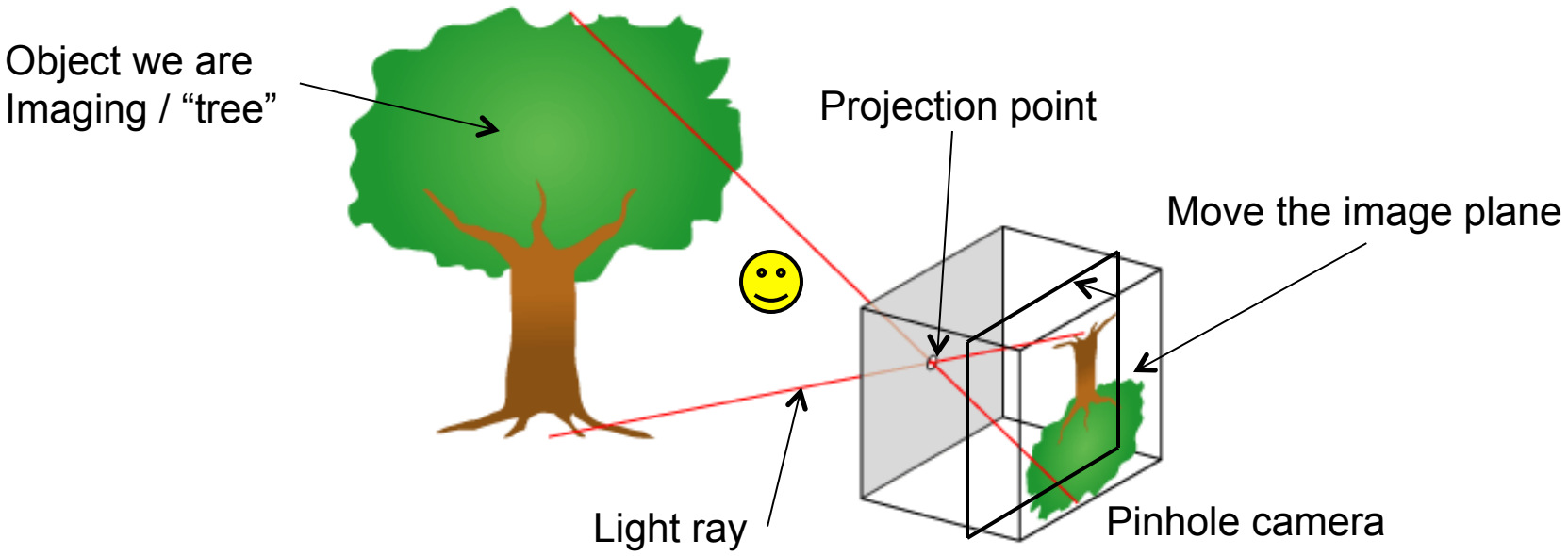
Camera Model

- As we move the image plane *closer* to the projection point, what would happen to the *object* we are imaging? ***It would get smaller.***



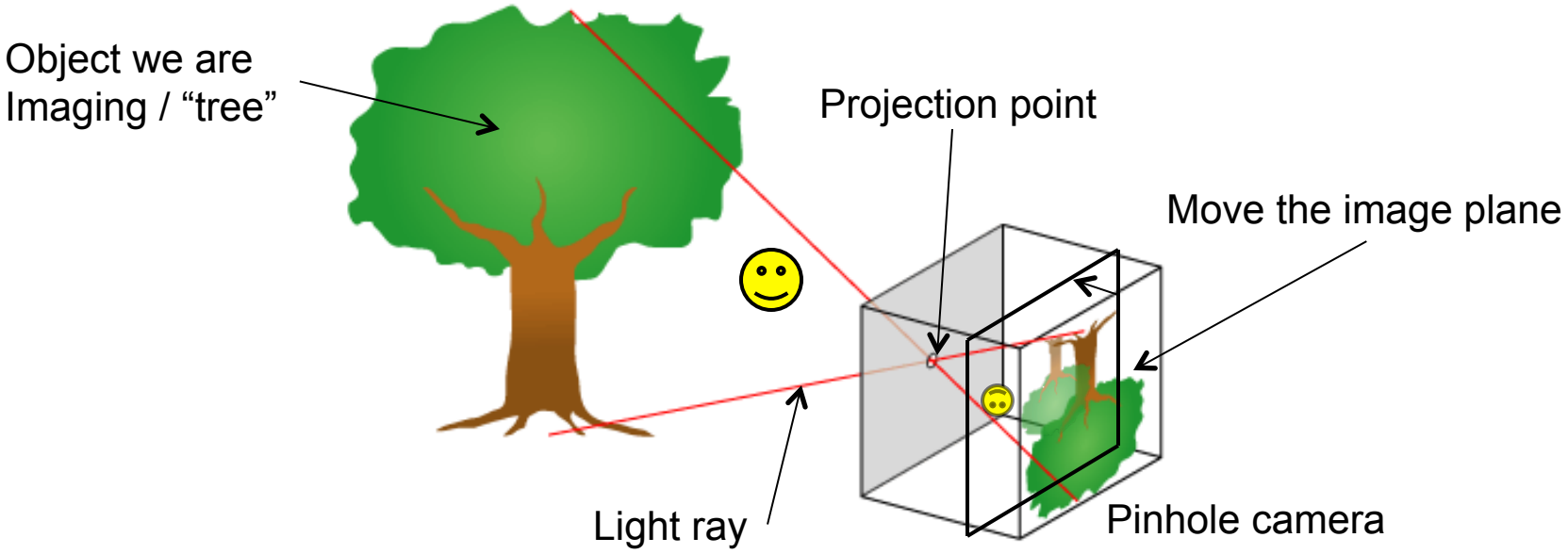
Camera Model

- What happens to the image that is formed, *overall*, as the image plane is moved closer to the projection point?



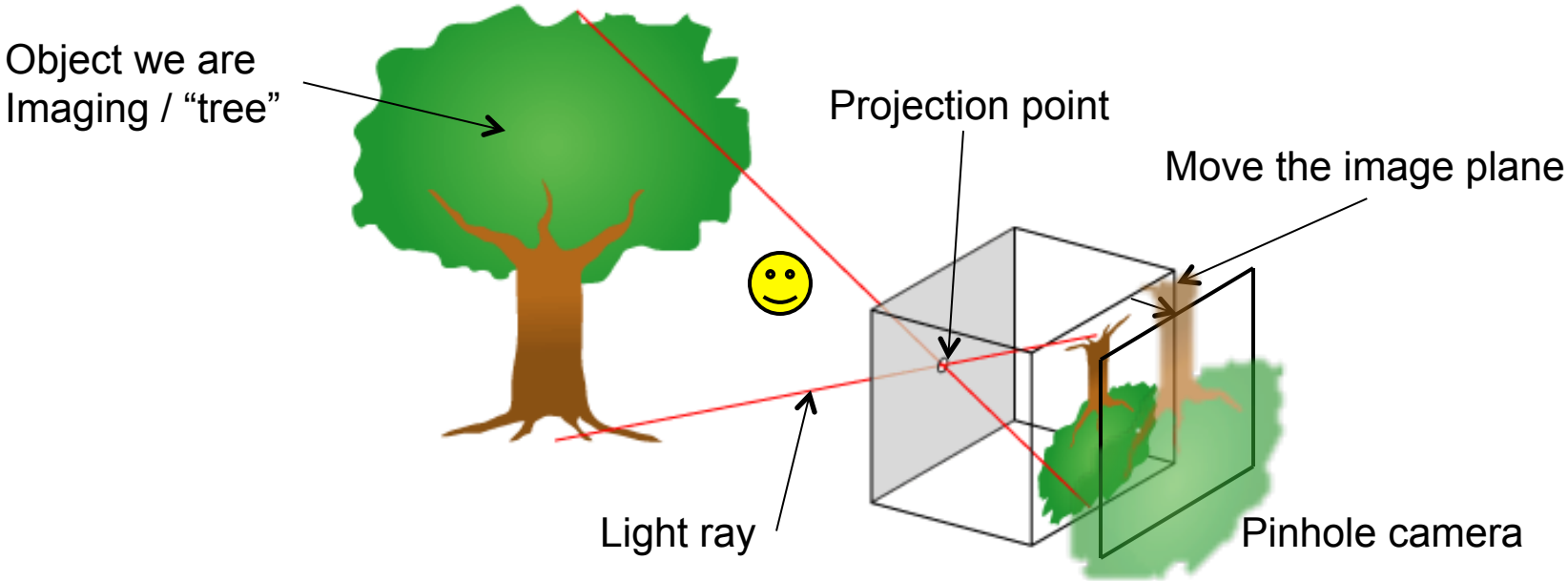
Camera Model

- What happens to the image that is formed, *overall*, as the image plane is moved closer to the projection point? ***It covers more area.***



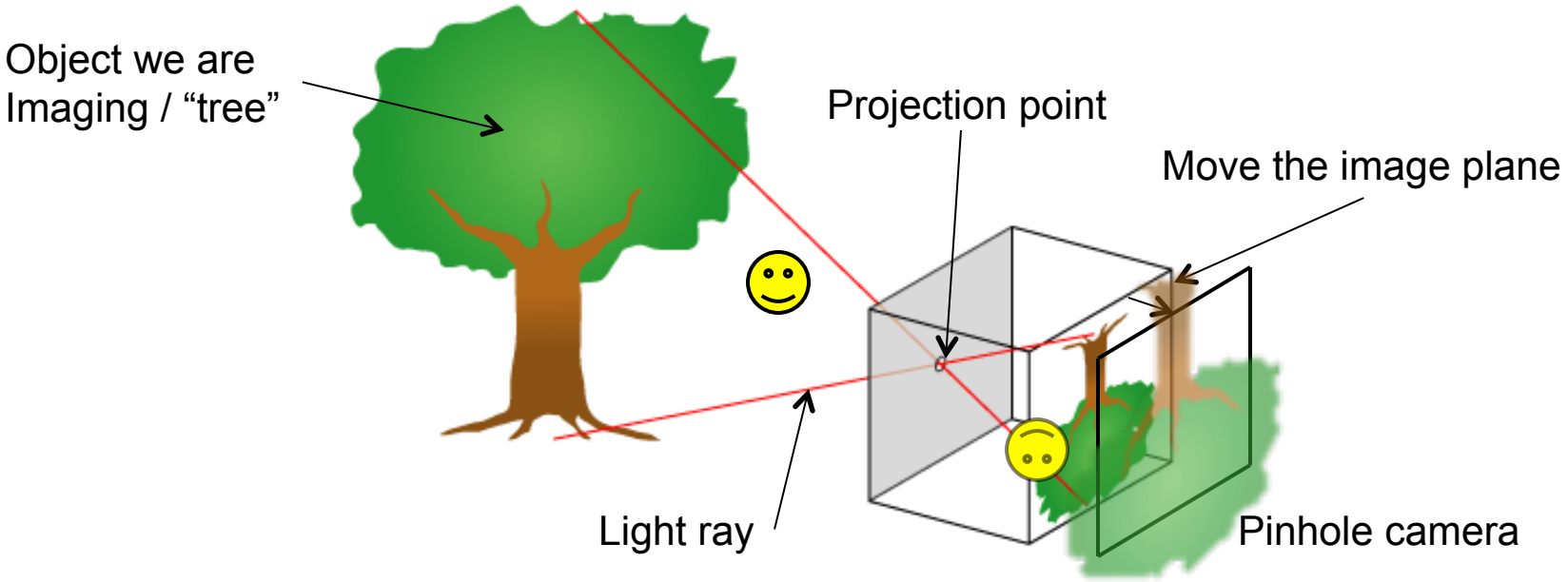
Camera Model

- Conversely, if we move the image plane *away* from the projection point the resulting object ***gets larger.***



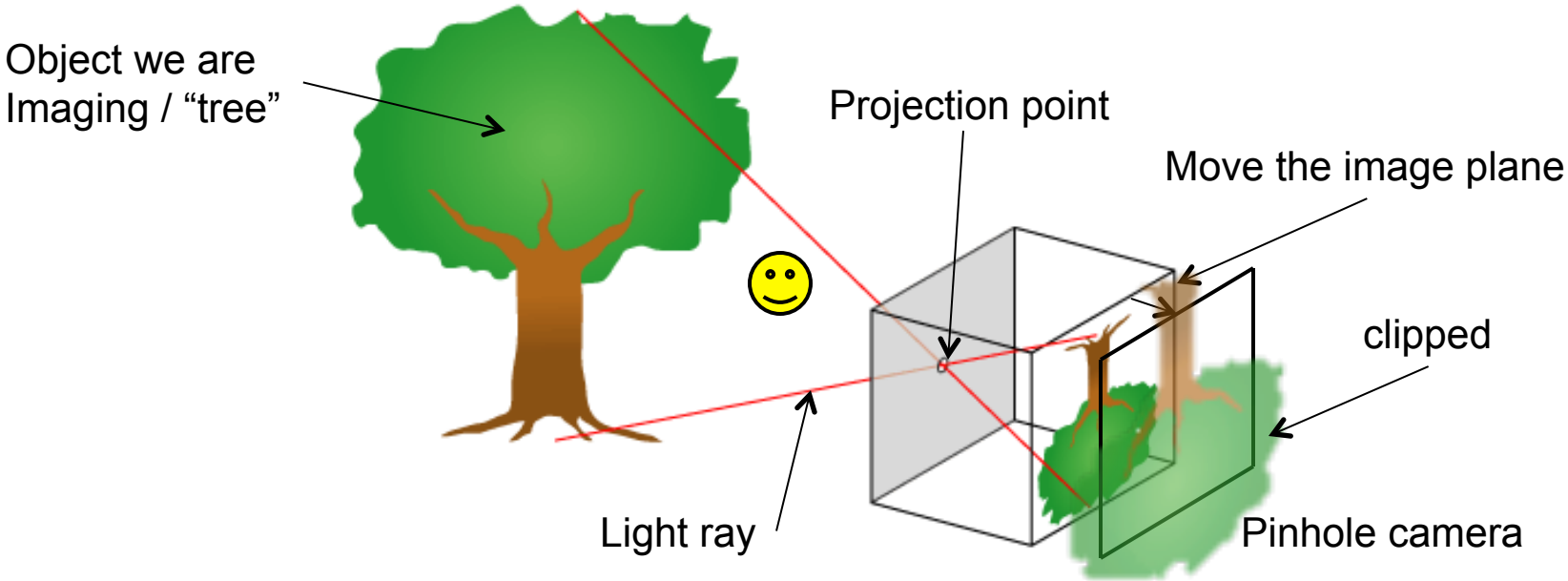
Camera Model

- However, because our image plane (film) is a fixed size we are able to capture less area of the scene.



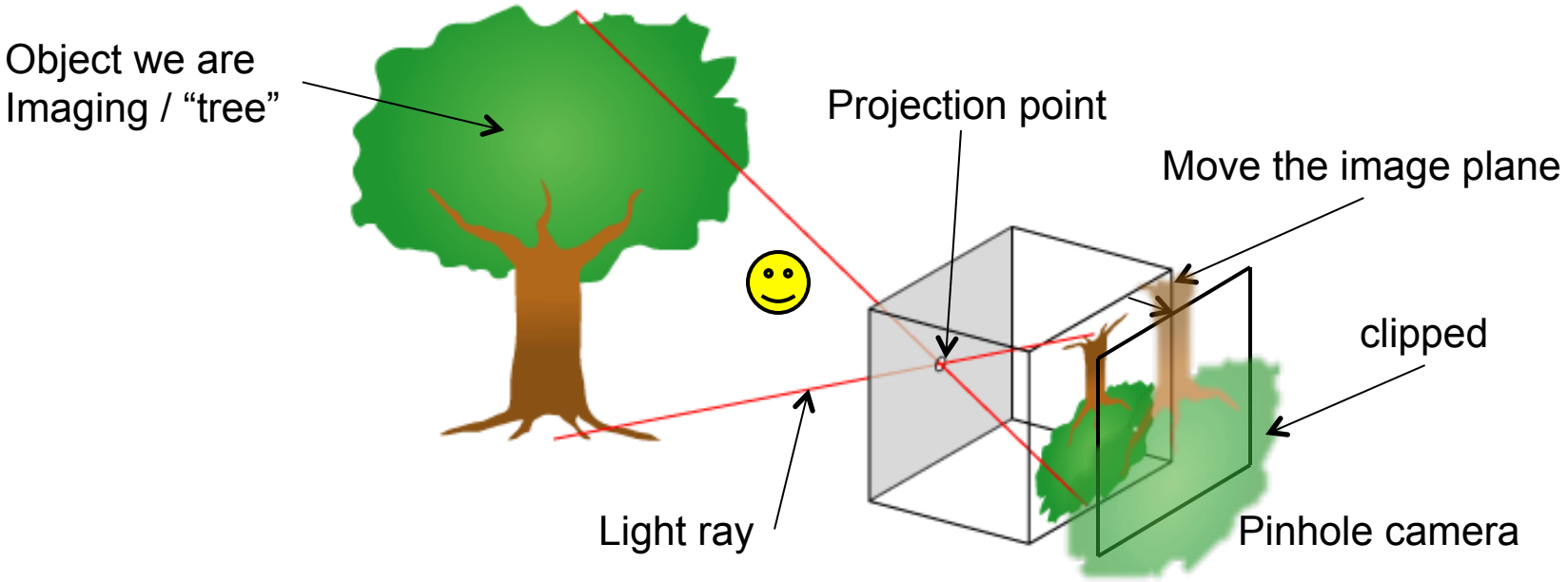
Camera Model

- In CG we call the process of eliminating the stuff that is out of bounds *clipping.*
  - *In this case, the edges of the tree are clipped from the film.*



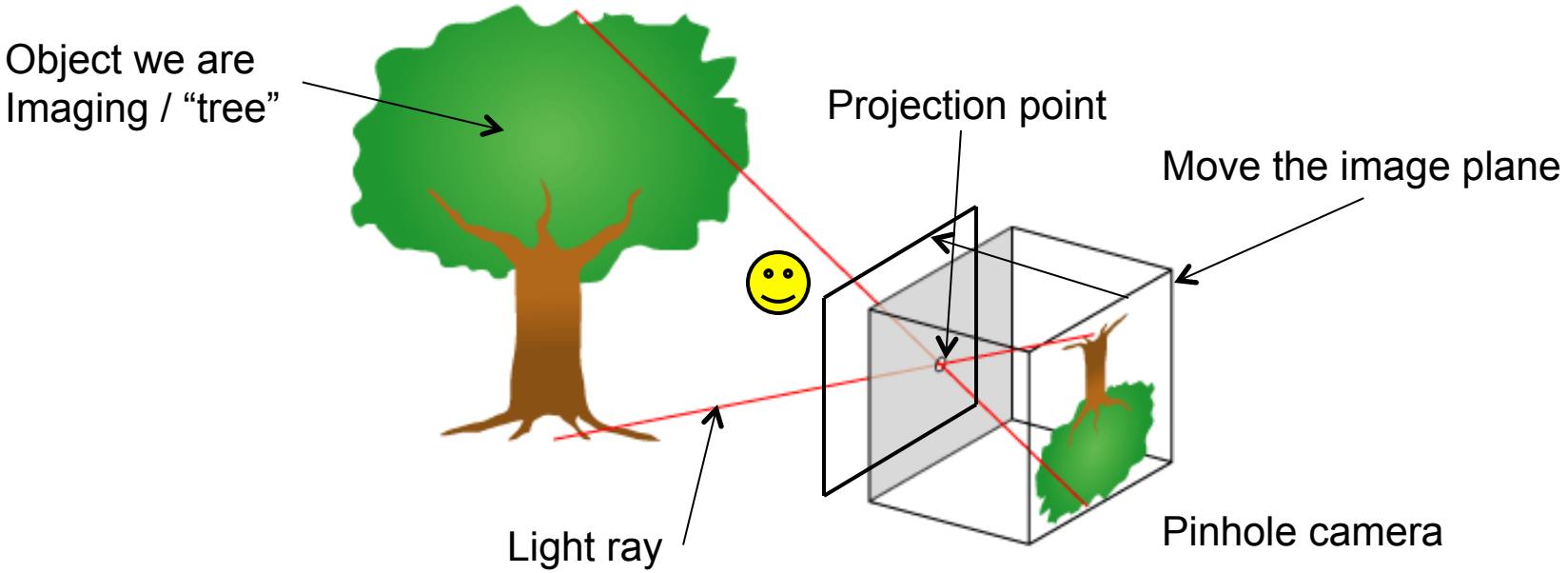
Camera Model

- In 2D we call it ***clipping.***
- In 3D we call it ***culling.***
  - We will spend more time on this later.



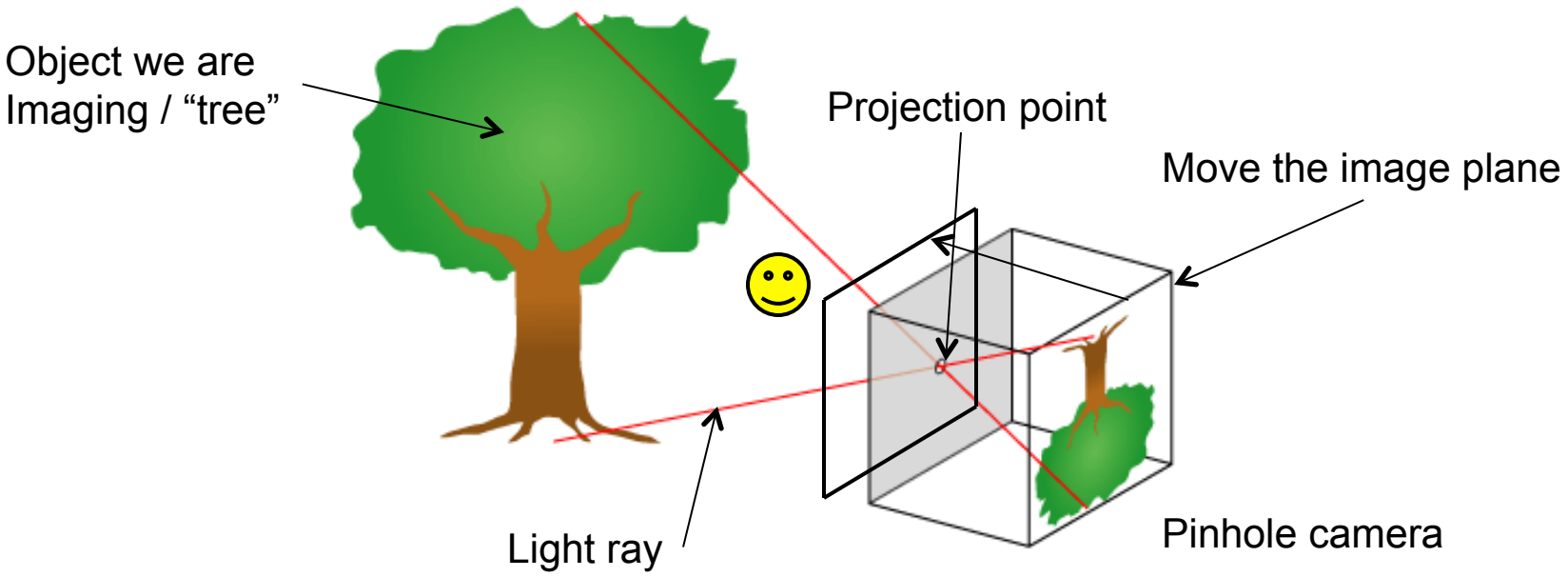
Camera Model

- What would happen if I moved the image plane ***in front of*** the projection point?
  - Can I do this? What would happen?



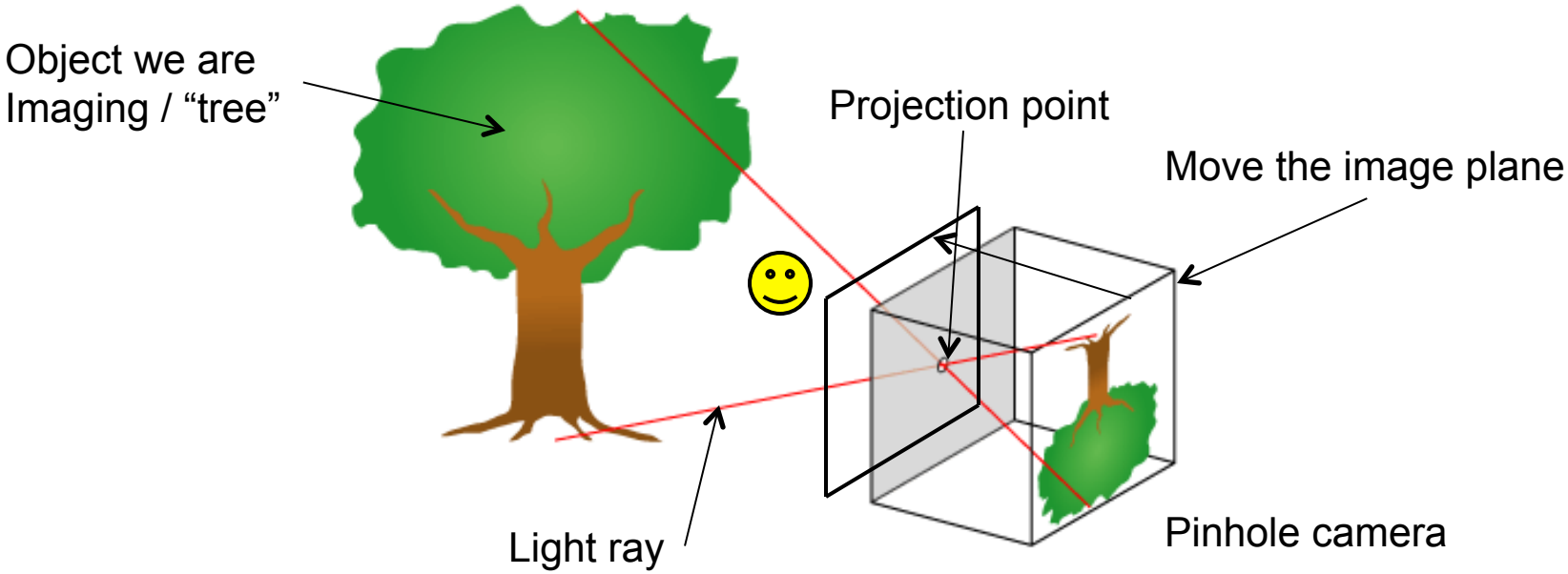
Camera Model

- Think carefully!
- Understanding this is integral to the class.



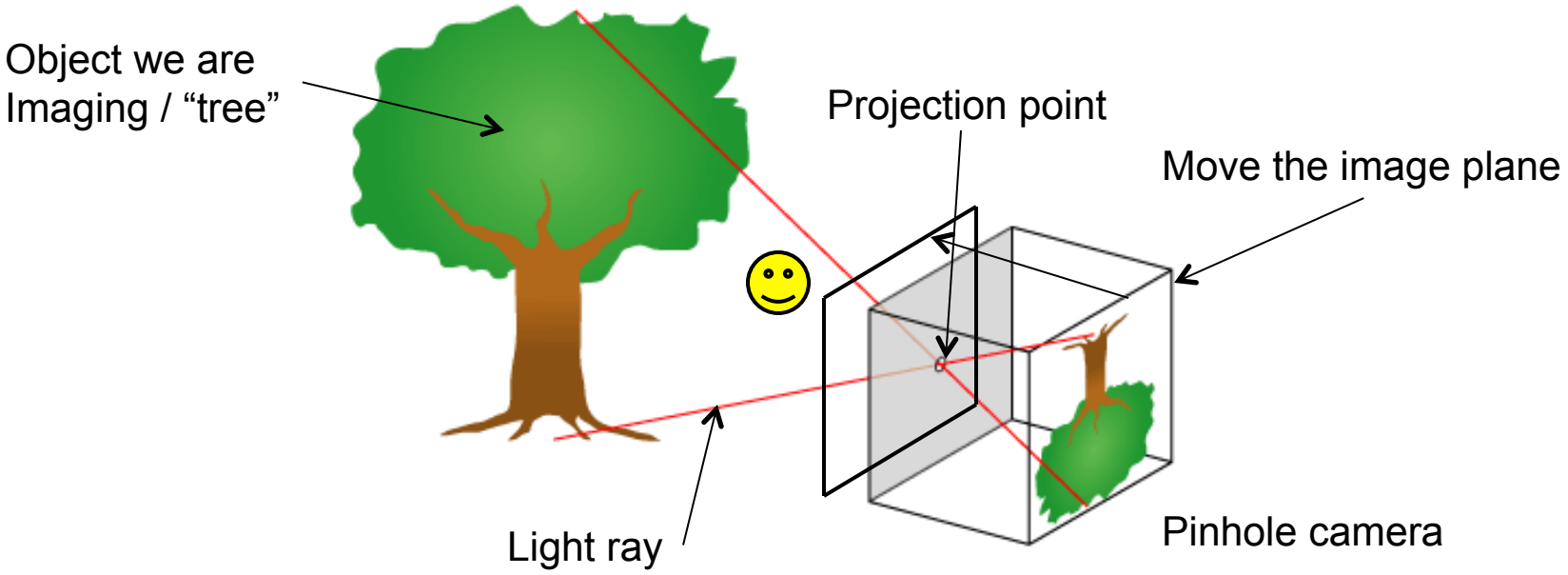
Camera Model

- Abstracting things (even more)
  - Pass a ray through the projection point and each point on the image plane.



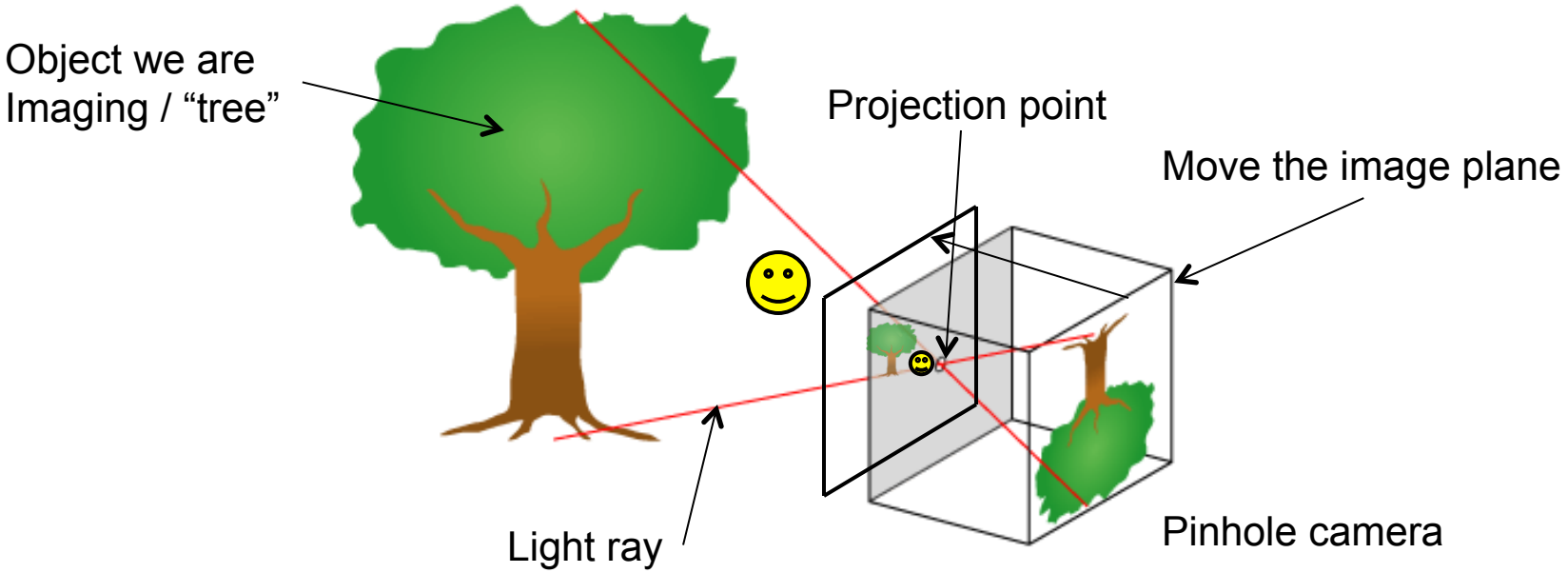
Camera Model

- What will the scene look like?
  - Will the tree be larger or smaller?
  - Is more or less captured on the image plane?
  - What else happens?



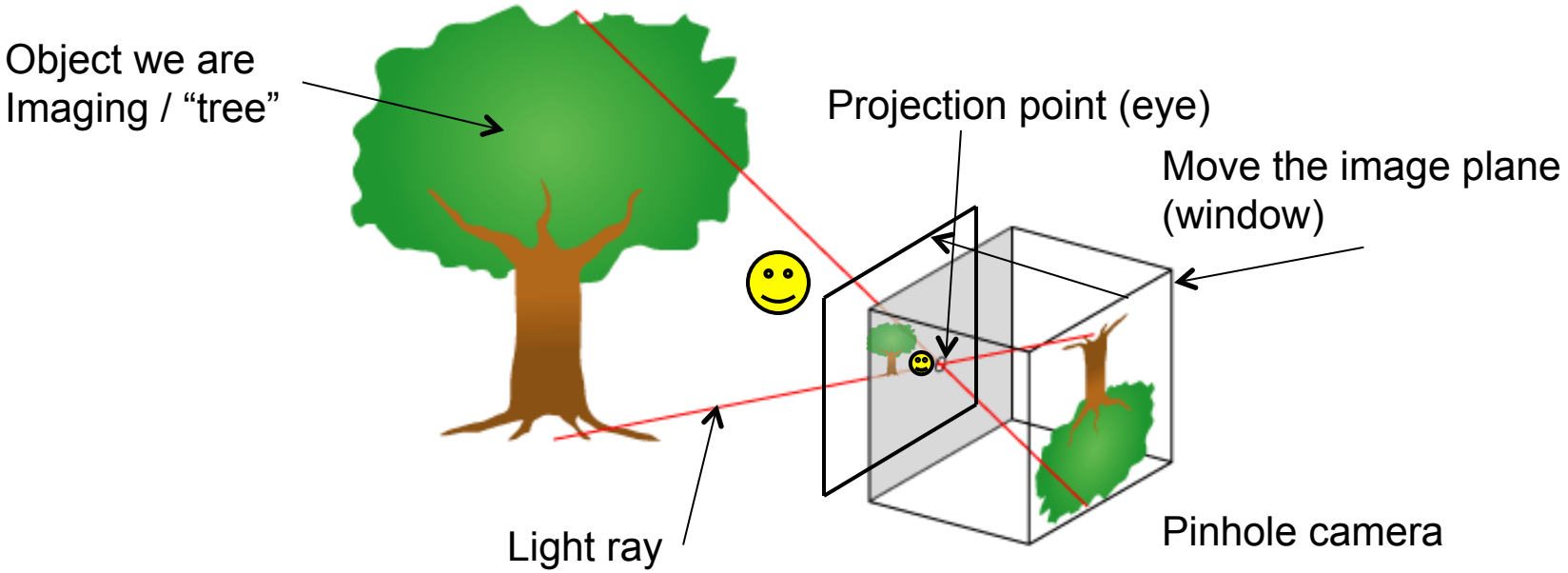
Camera Model

- What happens is…
  - Either depending on position of image plane.
  - Image is right side up!
    - In this example, image is smaller and covers more area.



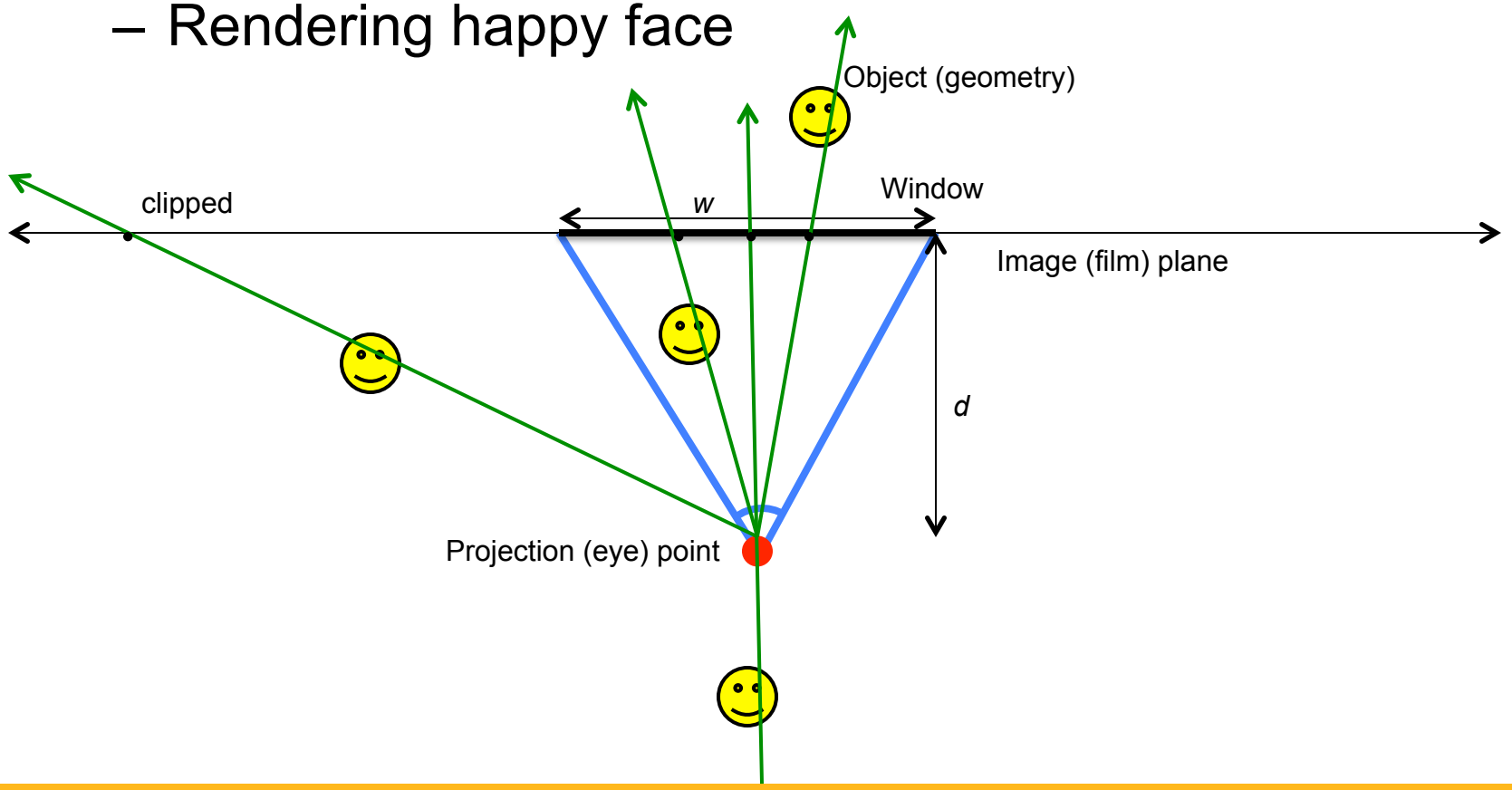
Camera Model

- Making sense?
  - We will be computing the projection (intersection) of an object (geometry) onto the image plane (window) using the projection point (eye), limiting what we see by clipping.



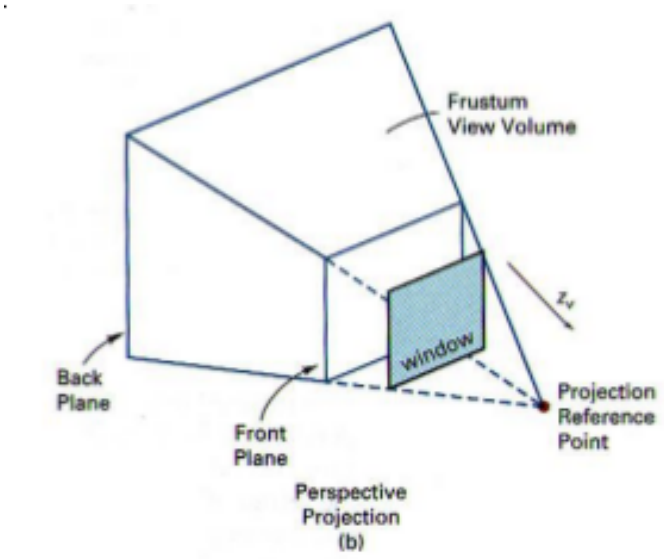
Camera Model

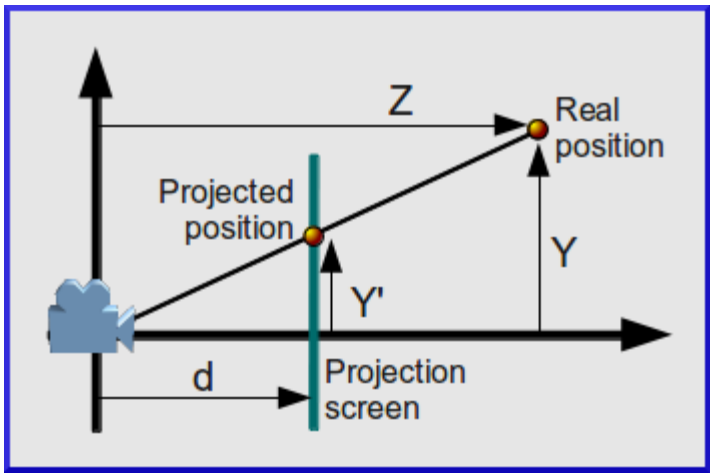
- Look at it this way…in 2D
  - Rendering happy face



Camera Model

- So, like a camera, but not exactly
  - We can do things cameras cannot do
  - Frustums help us limit what we consider for projection.





Camera Model

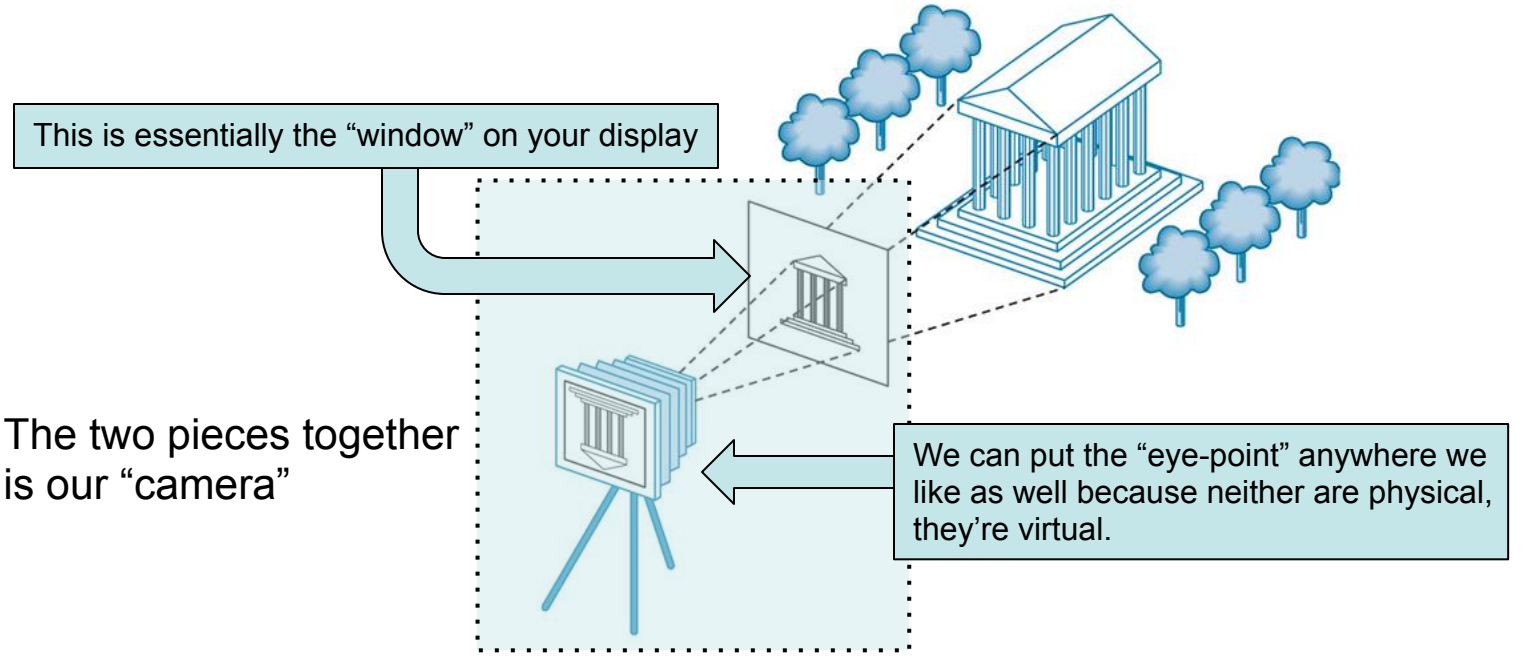
- Our camera, then, is defined by the relationship between the projection point (eye-point) and the location and size of a window we defined on the imaging plane.
- That window is typically what maps to our physical display.

Camera Model

- We can change either or both
  - The position of our camera system in space.
  - The parameters of our camera system.
    - Distance of eye to image plane
    - Position of window within image plane
    - Size of window within image plane
  - All of these will effect the image that is formed.

Camera Model

- Typically, the relationship would be something like this.



Next time…

- Get into the details
  - Actual code!
  - We will discuss these basic topics in relation to our first Assignment – which is available on CCLE
  - For next time you should read Ch.1 and Ch.2
  - You should also begin to get your development environment up and running with support for WebGL.

How hard is this going to be?

- Short Answer – hopefully not too hard.
- Let me show you an example
  - Written by a High School Student.
  - Worked with my group for eight weeks.
  - Did not know C or OpenGL beforehand.
  - Simulation and Visualization and Interaction
  - Multiprocessor enabled.