

## Computer Science 175

Introduction to Computer Graphics

[www.fas.harvard.edu/~lib175](http://www.fas.harvard.edu/~lib175)

time: m/w 4:00-5:30 pm

place:md 125

section times: tba

Instructor: Steven “shlomo” Gortler

[www.cs.harvard.edu/~sjg](http://www.cs.harvard.edu/~sjg)

[sjg@cs.harvard.edu](mailto:sjg@cs.harvard.edu)

office: md 243

teaching assistants:

Yuanchen Zhu

[yzhu@fas.harvard.edu](mailto:yzhu@fas.harvard.edu)

### textbooks

- unfortunately I haven’t found one that matches with much what I want to teach
- I will give out my notes
  - but these don’t have figures or full explanations
- there are some fragmentary related chapters on the web
- there are some reference manuals that most people have found unnecessary
  - “openGL programming guide, (sixth edition)” by openGL ARB et al
  - “openGL shading language, (second edition)” by randi rost
  - “the CG tutorial” by Fernando and Kilgard

### prerequisites

- ability to program in C (multihundred line program)
- familiarity linear algebra (we will quickly review)

### warnings

- you will need to use computers that support programmable shaders
- some of the assignments will build off each other
- much of the openGL gobbledy goop will be given as a black box.
- I expect to interact in class with the students, so I expect students to be engaged. (laptops...)
- I don’t capitalize or spell well.

### graded work

- 10 programming assignments
  - one weekers
  - hopefully due at mondays at 11:59pm
- unit per program
- 1 final project
  - 1.5 units
- participation
  - 1 unit or so

## late policy

- programs due on midnight
- up to one day late: -15%
- up to two days late: -30%
- after two days late: 0%
- you have 5 free days to use as you wish
- we will optimize late penalties/free days at the end.

## Standards

- you can work in pairs, or not
- Any help from other students must be limited and appropriately noted.

## platforms for class

- science center linux boxes
- windows
  - visual studio 2008
  - we can get copies for students (holloway@eecs.harvard.edu)
  - (cygwin/gcc if you can get it to work)
- mac
- must have recent generation graphics cards that support vertex and fragment shaders
  - first assignment will get this worked out

## lets look at the assignments

- 1: hello world openGL
- 2-3: 3d viewer
- 4: robots and picking
- 5-6: keyframe animator
- 7: meshes and subdivision
- 8: fur
- 9: under the hood: texture coordinate interpolation and reconstruction
- 10: ray tracer
- 11: final project

## applications:

- entertainment
  - video games
  - special effects
  - animation
- computer aided design
  - car bodies

- architectural environments
- visualization
  - medical scan imagery
  - fluid flow
- simulation
  - flight simulators
  - surgical simulators
- virtual world interactions

## syllabus

- we understand how to work on top of openGL
- we will understand what is happening inside of openGL

## topics

- getting started with openGL
- coordinate systems and transforms
- quaternions
- interpolation and splines for key frame animation
- overview of more advanced geometric modeling
- overview of more advanced computer animation
- subdivision surfaces
- color theory
- camera projections
- perspective correct interpolation
- viewport and rasterization
- reconstruction and filtering
- shading and shadows
- ray tracing and monte carlo

## class will follow the real time pipeline

- basic representation of object starts with a collection of triangles
- each triangle is described by 3 vertices
- each vertex is described by  $(x,y,z)$  coordinates
- a camera system is also described
- other information is attached to each vertex
  - color, normal vector
  - texture coordinates: pointers to places in images to be glued onto triangle.
- your program passes this information off to openGL

## vertex processing

- each vertex is passed through a *vertex shader* that you write and load into OpenGL.
- typically does “geometric” transformations on the vertex coordinates
- to place the objects correctly with respect to each other and the eye
- to get perspective effect

## fixed function

- 3 vertices for each triangle are collected
- triangles vertices are positioned within the window/viewport
- triangles are rasterized
  - which dots/samples/fragment/pixel are inside the triangle
  - data (color, normal, texture coordinates) are appropriately interpolated for each sample

## pixel processing

- information for each dot is sent through a *fragment shader* that you write and load into OpenGL
  - figures out final output color of the sample from the interpolated data
  - use normal for lighting/material equations
  - use texture coordinates to look up texels.

## final step

- decide whether to write into framebuffer
  - looks at data already there
  - main thing: depth test.