CS 174A Notes

W 1 T Lec 3-29-16

Matrix operation order:
translate -> rotate -> scale

Diana Ford
ford.diana@gmail.com

Computer Graphics

- Every single field, whether you are doing mechanical engineering, aerospace engineering, etc.
- If you are dealing with a computer screen and images on it, it is an important field to be in.
- You want that additional upper hand in your field on how to manipulate graphics generated data.

Applications of CG

- What is classified as the boundaries as a field at UCLA is different from other schools.
- Focus is on physics-based animations and physics-based simulations.
- Ford is the only one in the graphics lab pushing for game engineering.
- Utah for example is focusing on Virtual Reality in its domain.
- They are cancelling the Virtual Reality course for this quarter and it wasn't as important
- Nvidia called and there was an article about them partnering with the VR courses.
- Stanford has a Human-Computer Interaction lab and we don't recognize UI as going into the phase of graphics.

History

- 2000 B.C.
- Orthographic projection
- 1400s
- Perspective: Italian Renaissance
- Artists wanted to draw perspective and they wanted to know where to begin.
- Things can potentially converge from a distance
- 1600s
- coordinate systems: Descartes
- optics: Huygens
- calculus, physics, optics: Newton
- 1897 oscilloscope: Braun
- 1950-1970
- computers with vector displays

- 1966
- first true raster display
- How do we represent real-life images as real as possible?
- The field for the last 50 years is marked by how do we create realistic looking images and 3D objects.
- 1995
- Feature-length CG films (John Lasseter's Toy Story)
- LucasFilms was in trouble because he needed to pay off his wife and so George Lucas sold Pixar to Steve Jobs

Pixar Tin Toy (1988)

Genesis of Computer Graphics and Interactive Techniques
A PhD project at MIT in the early 1960s

- Ivan E. Sutherland, 1963

Quiz
1 Q. When was the term Computer Graphics first stated?
A. William Fetter of Boeing coins the term "computer graphics" for his human factors cockpit drawings 1960.
2 Q. When was the Graphical User Interface developed?
A. GUI developed by Xerox (Alan Kay) 1969
3 Q. When was Tron released?
Disney contracts Abel, III, MAGI, and DE to create computer graphics for the movie Tron released in 1981.

- How do we accurately position it in this world?
- Deal with matrices

Goals for the next 10 weeks

- Display things on the screen
- Position things on the screen

Next big trend in graphics is Robotics

- Create life-like images of humans.

What is an Image/Video?

- Array of pixels (one or more numbers)
- A video is a time sequence of images
- How are they formed:
- Objects in the world (static or dynamic)
- Illumination (light sources)
- Imaging device (eye, camera)
- We want to synthesize images/videos

Basic Elements

- Modeling
- How do we model (mathematically represent) objects?
- Solid works, Maya (mechanical engineering tools)
- Used to design things like plastic tools or even cars
- Rendering
- Modeling light in a virtual world
- Only approximated a solution when given examples rather than algorithms
- Interactions

Interaction
Input/Output Devices
Tools

- Modeling (use the skills that you learned and get creative and design a whole scene of your choice)
- Build a ray-tracer a bit later.

In previous offerings, there has been an emphasis on building an interaction within the 2nd project.

- Deal with user input, etc.
- Focus on understanding graphics and matrices as well as picking up enough OpenGL to learn projects.
- We get you to develop using OpenGL but we don't stress it or talk about it as much in class.
- Learn this in discussions and in the past, we could integrate it more into the course curriculum.
- Use this lecture to get the class to run this way.
- It helps to know the math.
- If you continue using mathematics, you will have "Aha!" moments when you see OpenGL or WebGL later on.
- Very simple basic algorithms that we come up with and generalize.
- They get confusing when we try to have high performance since they are possibly not even relevant later on.

Elements of CG
The graphics pipeline

- Modeling -> Animation -> Rendering

Modeling

- Representing the objects geometrically on a computer
- Plunk any UI on top and give it the same name.
- Smooth things so it looks more realistic

Rendering
Key elements

- Geri's Game

- Making the impact because they understand the algorithms and supplement it with a storyline that brings out the algorithm.
- Things look very soft on the eye, a lot of rendering and shadows
- We have the best way to integrate it in the story and the directory needed to sit and come up with ideas left and right.

Reflectance Modeling
- Three different types of materials and each reflects light differently
- Glossy, mirror light look all the way to diffused light

Texture
Pixar Shorts Collection Luxo Jr
- Pixar shorts are usually very soft and oversaturated.
- Give this feeling of warmth with rich colors.

Animation
Motion capture
- Import these into the game world and do animation in this way.
- File for running, jumping, eating, etc.
- This is really how graphics have become the standard in the gaming world

Fluid Simulation
Modeling
- Frozen algorithm for the snow

Animation
Behavioral animation
- Solve a lot of the problems of creating realistic graphics and now we have to create believable, realistic graphics.
- Just because it looks good can sometimes mean it looks too good.

Summary of the Syllabus
- Math
- Rendering
- Modeling
- Animation
- (Interaction)
- (Hardware)

Math
- Linear (vector/matrix) algebra
- Coordinate systems
- Geometry
- Points, lines, planes
- Affine transformations

- Projection transformations
- More geometry
- Curves, surfaces

Comments From Previous Courses
- A lot material
- Fast pace
- A lot of programming
- Tough third project
- Challenging final exam
- Great animation shows at the start of each lecture!

Class will be easier than you think!

Important Issues to Remember
- No plagiarism (of course)
- Do individual work
- Your code must work on HSSEAS systems

A Basic Graphics System
- Consists of input/output
- Computing & rendering system

Input Devices
- Keyboard
- Mouse
- Light Pen
- Game controller
- Tablet
- Data glove
- Other sensors

Full body tracking: Tracks depth and you are really immersed because now you can walk

Output Devices
Display
- CRT, Plasma, LCD, Micromirror
- Pixels that are emitting light pixels
- You can adjust the intensity of what is displayed
- If you have a voltage and a liquid crystal, you treat them together and you can have them look normal and test for it by going to the light and turning it black.

Printer
- 2D and 3D Printers

Plotter

- Our monitors are non-linear, meaning we sometimes have to change the gamma values

Standard Display Devices
- CRT (Cathode Ray Tube)
- Shoots lights and fill in pixels in a non-linear manner
- LCD (Liquid Crystal Display)
- Plasma

Exotic Display Devices
- Immersive
- Augmented reality
- We can have head-mounted displays with views of 110 degrees.

Analog Video Signal Format
- Not perfect in terms of mapping

Basic Analog Display Architecture
- Grey scale: 8 bits/pixel
- We can have three bytes (24 bits) for RGB (red green blue)

Images - Monochrome
- How many intensities are enough?

Rendering System
Software
- Use OpenGL API so we can communicate using the graphics pipeline.
- Figure out how to communicate with the graphics card and use one chunk and code everything.
- As graphics cards improve, there started to be a need for different sections as we send out different vertices separately.
- Display on a 2D screen separately and figure out how to display the images in it.
- Determine the positions of the image in terms of 3D models.

Unreal Engine 4 - Kite Trailer
- The kite was released in last year's GPU conference, and up until then, 3D real time animation was not possible.
- Nowadays, we have 60 frames per second and now we have produced something of higher quality.

Graphics Pipeline
- Modeling
- Illumination
- Viewing (Projection)
- Clipping

- Visibility
- Rasterization

Modeling
Geometric Primitives

- Points
- Lines
- Planes
- Polygons
- Parametric surfaces
- Implicit surfaces
- Etc.

Easiest way to debug graphics is to display an image on the screen!

Illumination

- Best algorithms to eliminate things is smooth elimination
- Interpolate half the triangle with the red and use an algorithm to interpolate half the triangle with the green.

Per Pixel / Fragment Operations

W 1 R Lec 3-31-16

- Get the enemy to communicate and the NPC can continue and look a little more intelligent.
- Dot product
- Returns a scalar
- Cross product cannot return a 2D value
- Returns a vector
- Returns the area of this parallelogram
- Graphics is based on the right-hand rule
- You need to manage the coordinate systems and be able to perform the correct transformations
- Orthonormal is usually what we are taught in high school, so we are used to this.
- Normalize and then divide everything to give it the right position

Perspective

- When we start dealing with perspective, we want to be able to handle it accurately.

Dot Product and Perpendicularity
From Property 5:

- $a * b > 0$
- $a * b = 0$
- $a * b < 0$

Linear dependent: Multiples of each other

- To describe a plane, you need separate vectors; otherwise, you will get an arrow
- When you are scaling a chair, you are NOT changing out its shape.
- You are only changing the ratio of height and width.
- If you have a 3D model and you would like to look for other 3D models, you will be in a position where you can say which 3D model is equivalent to another 3D model.
- Doesn't change perspective, only changes direction!
- To calculate this, grab the matrix and decompose it to 2 other matrices.

W 1 Dis 4-1-16
TA: Garett Ridge
garett@cs.ucla.edu

Part I: Course Logistics

- Projects, Textbook, etc.

C++ and WebGL (JavaScript)

- Both of these use OpenGL, an interface into your graphics card
- WebGL = javascript + OpenGL calls
- C++ plus OpenGL calls, on the other hand, sometimes just called OpenGL

JavaScript

- Functional programming language
- More in common with Lisp than in Java
- Not even related to Java and it is NOT a child of the Java language.
- Most lines you'll type for graphics commands look the same in both languages.
- Your work is lots of copying and pasting commands instead of typing
- You don't have to be an expert in JavaScript to start on Project 1
- When you're not copying and pasting, you can look through the template to see how the code works.

Assignment Zero: Set up your environment

- Most people are more familiar with C++ because of CS 31 and 32
- Starting with it will thus be our gentlest way to jump into graphics programming
- Assignment Zero: Try it out to get used to how a graphics project looks like.
- See it change something in the output visually.

Which textbook should I get?

- "Interactive Computer Graphics" by Edward Angel

- 6th edition is in C++ and there's a PDF online.
- The 7th edition is in JavaScript
- JavaScript handles a little bit easier than the C++ version of OpenGL.
- Modern web browsers have a built-in graphics debugger that can deliver messages your C++ compiler can't
- Less typing / some saved steps
- Your code will be integratabtle with websites and the web, things that are here to stay.
- You want to be able to do graphics on the Internet for interviews.
- No setup required
- Runs from a file immediately
- Edit each file directly
- The author's demos can all run online in JavaScript.

Assignment 1: Animate a bee

- Copy a certain existing animation as closely as you can
- Place cubes and spheres correctly
- Use JavaScript
- Mainly, you'll be graded:
- On including all the pieces of the scene (bee, flower, ground)
- On the shapes pivoting off one another at the proper connection points
- Emphasis on grading is on the hinge points than color scheme, etc.
- No bonus points for making it really cool, but it looks better if it is customized.

Assignment 2: Animate anything

- Make any animation or game as long as it includes all the required techniques
- You'll be graded on creativity, complexity, and quality
- A video of your animation will e voted on by the class — the best animations receive lots of extra credit towards the course grade
- Online showcases of prior animations exist; yours will join them since we're doing it in JavaScript.

http://web.cs.ucla.edu/~garett/cs174a_projects2/

- You have to use the raw OpenGL calls as your base model, so it is hard to integrate things like 3.js or other game engines.

Assignment 3: Trace Rays through our scenes

- We give you a full description of a simple scene (in a text file)
- You'll build an image by mathematically following rays off right that bounce around the objects in the scene
- You'll be graded on implementing all the features (reflection, shadows, lighting) and correctness of images

Total available grades are 120% and then she will bring it down to 100%, so quizzes are extra credit

- The quizzes could potentially very difficult
- The quizzes should be actually doable and able to score lots of points on.

We'll make shapes out of math.

- Remember your high school teachers making you do it with graphing calculators? This will be the more advanced version of that.

We aren't using simple functions; we will be using tough functions to mathematically represent.

Discretization

- Break it up into discrete pieces instead of continuous types of shapes that you have to use calculus on.
- We don't know how to tell a computer to draw most shapes because of their complicated non-linear formulas.
- Break them up into a finite # of line segments between discrete points.
- Evaluate the function at different values.
- For a shape like this, if we solve it for various values of the parameters, you will get x, y, and z points.
- Do trig on it and we know the diagonal's length is 1
- cos(theta) = x/r and sin(theta) = y/r

Triangles

- Simplex: simplest shape you can make without changing the dimensions
- Just references the other list of all the point positions
- All that math we did to compute the endpoints can be used for the list of points

Two methods of delineating triangles

0. Exact coordinate points
0. Indices

draw_elements and draw_arrays take in various parameters

- You want your info as compact as possible so it can stay in the cache
- You are more likely to revisit a point if the triangle includes the index to that point.
- Points take up a lot of space to write.
- It will be x, y, z and the other vectors.
- All these other things don't want to be duplicated since you want the index to fit into the CPU cache rather than the entire vertex data out.

Another exercise:

- Interpolation: List some points along the line from one point to another
- alpha in the range 0…1

Generated point = (1 - alpha) * P1 + alpha * P2

- Works for each coordinate, apply it to X, Y, Z separately.
- Parametric equation: as we increase a single parameter, we generate brand new points.
- We could have several if it is a surface instead of a line segment.
- For a filled in circle, maybe we have two parameters and it takes us towards the top of a circle.

What's a matrix?

Elementary Matrices

- An elementary matrix E is an n x n matrix that can be obtained form the identity matrix I_n

Rotation Matrices

- A rotation is two shears
- But certain properties have to hold. This one is NOT a pure rotation matrix.
- Columns have to be perpendicular.

Translation Matrices

- Moving a shape over to a different place.
- Every matrix applied to x, y, and z are always just stretched to warp around the origin.
- The reason they did that is because we are applying x, y, and z
- No matter what matrix we put in there, it will always come up with 0 for the origin point.
- Land right back at the origin and map onto itself.
- Take in a little bit of x, y, and z around.
- Toss in a constant for this!
- Affine interpolation: the line segment won't be changed after all the transformations
- Warping things around somewhere else beside the origin.

3D translations are the reason we use 4x4 matrices instead of 3x3

Midterm:

- Matrices on the midterm will be simple enough to do math on and see where the shape lands.

Matrix Order

- The hardest concept in the class
- Matrices are NOT commutative
- You cannot simple switch around their order.
- You cannot move the matrices but you can evaluate them in whatever order you like.

- Evaluate some pair and turn it into some matrix.
- Do it in some silly order and eventually you come up with the right answer.
- Using a fixed coordinate system that isn't going anywhere.
- It is always right here and any rotation goes around the origin.
- Keep adding in new pieces from the right side, but add values from left to right.
- Two approaches. Multiple starting from:
- Right to left
- Left to right

W 2 T Lec 4-5-16
Transformations - 2D

- Translate (W): Translating their position by rotating and scaling them
- Made slightly easier if you have game engine interfaces where you can do method calls, or you can use the GUI interface.
- Adjust the arrows and drag the 3D object and move it across the screen.
- Find the mathematical basis of matrices and you will get to do the one method call to create these translate, rotate, and scale objects.
- Some of the things that are emphasized will be tested on in the final and the midterm.

How Do We Draw Objects?
Z-buffer

- 3D modeling displayed on a 2D screen.
- A couple of algorithms where we have a whole lesson on them

Raytracing Algorithm
for each pixel on screen
    determine ray from eye through pixel
    find closest intersection of ray with an object
    cast off light
….

Z-Buffer Algorithm

- Stores objects, pixels in relation to distance.
- Closer to the screen == higher #
- Highest # gets displayed for that pixel
- These algorithms take into account the eye and the viewing position.
- When the eye position changes, depending on how close you are this makes a difference in terms of transforming things and representing them accurately.
- In some instances, take into viewing direction.

Z-Buffer Graphics Pipeline

- This is where we rotate and translate all the 3D objects -> then we move to the camera view and there will be a series of transformations in the pipeline

- Translating cameras and then we go into projection transformations
- Represent the image accurately from the correct viewpoint and go to clipping and get rid of things we don't need.
- Prospective division and viewport transformation.
- This is where we have a 3D world and we have a window into the 3D world and we want to stare outside the window and now we focus a viewport and where do your eyes land.
- A way to eliminate the window and not pay attention to everything.
- How do we go about going to this viewport?
- Rasterization: Displaying pixels correctly for color and for lines.
- Many transformations and this is just a summary

2D Affine Transformations

- Preserve the angles of the 3D object and if you want to transform this chair, you aren't going to make all the legs shorter.

Post-multiplication

- Stick the matrix transformation before, rather than after

Solid body transformation

- Distance and angles are invariant

Identity Matrix Transformation

- $a = d = 1, b = c = 0 \Rightarrow x' = x, y' = y$

Scaling Transformation
$b=0, c=0 \Rightarrow x'=a.x, y'=d.y$

a and d are known as Scale factors S_x and S_y

If $a = d > 1$, we have **uniform** enlargement

If $0 < a=d < 1$, we have **uniform** compression

Reflection Transformations

- Reflections will be asked on both the midterm and final!
- Flip primitives and if you flip them wrong, it will take a chunk of your grade.

Special cases of reflection

- Two other cases

0. Reflection across X
0. Reflection across -X

- Go through on your own time and do practice!

Shearing

- Think of a deck of card stacked on top of each other and pushing the books.
- Often, big books have hard pages.
- See the pages shear.

Rotation

- We start out fairly simple, and we just replace the values, stuff them in, and rearrange the equation to come up with the matrix
- You don't have to memorize this matrix, just know this is what the rotation matrix looks like

Special cases of rotation

- Go through these on your own time and make sure you know how to rotate a triangle
- NOT going to be explicitly like this on the exam but Ford wants us to visualize it.
- Figure out which way is clockwise and which way is counter clockwise.

Translation

- Slightly different from what we are used to.
- Start noticing that these positional representation of a square and we are multiplying it with an abcd that is going to affect its translation
- Depending on where the point is, you end up with different shapes -> thus, you don't always get an accurate representation.
- Start noticing a basis change and we will talk about it in one second.
- Location is shifting and the shape shifts as well.

Cameras (and the Eye)

- There is also different types of technologies i.e. cameras and lenses with different depths of feel that also affect things.

General Transform Matrix

- Go through the abcd matrix and we can scale, rotate, and shear with it.
- We can use one matrix to perform all these types of actions
- How about translations?
- There is no space for translation! Math is all about generalizing things, so what is the one matrix that can provide a translation.

Points vs. Vectors
Convention

- Vectors don't have a location, so they can take a 0
- Points need to be normalized and have a 1 to describe a point
- Every object is in 3D so we want an x, y, z-component

Switching Representations

- Homogeneous to normal:

- Vector: remove 4th coordinate (0)
- Point: remove 4th coordinate (1)
- Homogeneous systems are ways to accurately describe and protect against pests or attackers

Q. Is 1 the standard distance?
A. Normalized everything else in math
1 is always standardized for math

Homogenous equations exist if we have a distinct notation in English
- A vector can represented a point minus 1

Relationship Between Points and Vectors
- A difference between 2 points is a vector:

Q - P = V
- We consider a point as a base point + vector offset

Q = P + V

Coordinate Systems
- Perception of a building while in a moving car
- Your position is going to be different, so you have to use a different coordinate system while the car is moving.
- As the coordinate system keeps changing, that can be used in the coordinate system for the building.
- (v_1, v_2, v_3)
- **Basis of assignment 1**
- What it comes down to is actually starting assignment 1 until you have an “aha moment” with where you multiple this matrices!
- Talk bout changing the basis for robot arm’s and calculating the drawing angle.

Homogeneous Coordinates
- Take into projection and viewing direction
- Get a matrix with a 3x3 matrix
- * The last row puts the matrix in a homogeneous coordinate system
- t_x and t_y are responsible for transformation
- Come up with #’s rather than generic letters and then multiple it by x, y, w
- Make sure you have two sets and t_x and t_y will be moved.
- End up with a transformation for sure.
- Make sure to play around with t_x and t_y to get an accurate translation!

General Purpose 2D Transformation Matrix
- Usually find these at 0 rather than randomly replacing 0’s
- If you want to do with pre-multiplication
- If p = 5 and q = 6, you would try to find the same result for this matrix.

Composite transformations

- Translate it by T_2 and p", multiply it by p"' and for efficiency's sake, multiply out all the matrices together first and then multiply by the point and get the same result.
- Because we are dealing with graphics cards, we want to be dealing with T_3 * p

Composite rotations

- With rotations, all we need to do is add the values
- Efficient way of going about it rather than T_1 by p and T_2 by p.

Transformation - object not at origin

- Common mistake in Animation 101 or Game Engines 101 is transforming the character and this is Week 1 or 2 of Game Engines course
- We have a character and we want this character to go to that house and all of a sudden, you want it to face another character.
- The origin point is the 3D-world and it has an origin that holds the actual object.
- You get into this predicament and the solution is to stick the character into an empty game object that has the local coordinate system and it is 000.
- Perform calculations the way you did so it doesn't have issues.

Rotation example of a triangle at an arbitrary point in space - 2 -

- Triangle translation to (0, 0, 0)
- Rotate
- Triangle translated back to original point

Matrix multiplication is done Right to Left (this is the most computationally efficient way!)

- We are going right to left to multiply these matrices
- As we know, if we have A B C, and we are going right to left, the order of execution is C * B * A

A B C
(A B) C
A (B C)

- All of these produce the equivalent answer!

We need a situation where we know which is wrong and we can say why.

- In this situation, we are multiplying from Right to Left
- The first matrix that gets multiplied is T_3 and since it is NOT, it is the wrong answer on the multiple choice.
- We are dealing with modeling transformations and we are moving them around in 3D space.
- We are still stuck in transforming the 3D objects.

Scaling about an arbitrary point P in space

- Generic view of the general matrix form

Reflection through an arbitrary line

- Important to get your head around this
- Lots of reflection questions because they can get confusing.
- ** You reflect an object in respect to a line, so you must make the line in line with one of those lines, to do this you need to rotate **
- If you have this triangle and you want to reflect it on y, it is kind of sitting

Go home and test this out with one example to understand real matrices

Window-Viewport Transformation visualized

- Create an accurate viewport representation of the world/window
- This is a surprisingly difficult concept to wrap your head around and you need to scale and rotate to get back to the origin.
- The point is that everything has a coordinate system.
- In this instance, we are using the Cartesian coordinate system and bringing everything back to the origin.

W 2 R Lec 4-7-16

- 3-dimensions have a 4th variable
- Do some work on coordinate spaces and do some exercises where we go through a couple of slides with a partner next to you.

Homogeneous representation of a point in 3D space
P = | x y z w |^T

Affine Transformations in 3D

- General form of a homogeneous matrix is 4 x 4
- The rotation, scaling, shear is done on the left end, and the right end is reserved for scaling

Scaling Around the Origin

- If they are different sizes, it scales non-uniformly

Shear Around the Origin

- Along x-axis

3D Rotation

- With rotation, in order to perform correct rotations, we need to decompose rotations into x-roll, y-roll, and z-roll

Counter-clockwise is always positive

Three Axes to Rotate Around

- Translate it back and rotating around the origin.
- Take all three axes into account and have an accurate rotation.

- One of them is called the **nimble lock**

Z-Roll

- If you add another row for z, then P_z will affect x and y.

Rotating around the y-axis

- y is not moving but the x and z values are getting larger.

Locking any of the axes

- The other two axes will be affected while the locked axis is the one spinning around.

Inversion of Transformations

Inverse of Rotations

- Transpose of that matrix and pure rotation only
- No scaling or shearing involved
- See what these general rotations look like.

General 3D transformation matrix

- Make it similar to what each slide had
- Group them in a way to potentially see them in math (text or paper) where they are referred to as a T-K matrix

3D reflection about a **plane**

- the -1 value reflects on is the value not included on the “name” of the plane
- xy plane has -1 for z
- yz plane has -1 for x
- xz plane has -1 for y
- **… You get the point!**

General rotation form

- Rotate around the x-axis: the x-axis doesn’t change
- This means the axis we are rotating around has a value of 1
- If rotating around the x-axis, then x = 1
- If rotating around the y-axis, then y = 1
- If rotating around the z-axis, then z = 1

Positive rotations

- If you rotate around an axis, the axis being rotated around does NOT have its value changed.

Rigid Body Transformation

- Affine transformation
- You could have a deck of cards that you can shear

Composite 3D Rotation About the Origin

- "Euler" Rotations
- Three types of transformations:
- Going around the x-axis
- Going around the y-axis
- Going around the z-axis

Rotation Around an Arbitrary Axis

- We want to perform a rotation theta degrees, about an axis in space passing through he point and we have its direction cosines

We have a general matrix and then we can perform the derived general form for rotating around x and y.

- Make sure your vector is coincident before doing anything else.

Special cases of rotation

- Happens when the axis is parallel to the x, y, or z axis
- We are rotating around an arbitrary axis in this situation

Derivation

- Visualize what is going on in this slide
- All this slide does is go through general derivations of general matrix form
- Picture a random coffee cup in a 3D-virtual world and we want to rotate it just because.

Preservations of Lines and Planes

- To prove the lines are preserved, we just multiply it out by any M and by process of expanding out basic algebra, we should end up with our proof.
- Read slides 62 and 63 at home.
- Ask questions in discussions.

Transformations of Coordinate Systems

- Coordinate systems consist of vectors and this provides an alternative way to think of transformations as changes of coordinate systems.

Robot Arm

- We have a piece and we want to stick it at the top.
- We have instructions and we want to translate y by height
- The point of this is to multiple from right to left and calculate all of the upper part to calculate the matrix transformation.
- L -> R
- Kept changing the coordinate systems for each one, so to build the matrix,
- Coordinate system change
- Calculate the final location position

W 2 Dis (Theresa) 4-8-16

Part I: Review

Matrices and Rotations

- Define a coordinate system
- When you multiply it out, you get a new vector
- The whole reason to do this is now, you give yourself a new variable in a new coordinate system
- After you are done, to get back to the original, all you have to do is the inverse
- Graphics will be all over the screen, so it will be easier to perform operations if you go back to the original each time.
- Easier to work with a coordinate system that is easier to multiply
- You would still get the same exact result if you translate back and it's computationally easier.
- In graphics and OpenGL, they do this by default in order to get somewhere.
- A live demo in JavaScript
- **This logic is fundamentally important!**

Homogeneous matrix

- If I am working in 3-D, I am working in a 4x4 matrix
- If you have a homogeneous coordinate, you can parametrize and add a new variable to shift and translate each axis.
- If you don't have this extra dimension, you cannot translate off the origin!
- The mathematical reason is that if you don't have this, you cannot distinguish between points in vectors and you cannot project anything.
- If you want to project something in 3-D, you would have to go up one level or else you will lose the third dimension.
- We are still seeing a projection on 2-D!

Q. Why can't we directly rotate from the matrix instead of going back to the origin
A. You could but it is much harder. You would have to unrotated back to the original, then rotate with alpha + beta.

- You could also tilt the whole world, but I don't want to imagine how that is mathematically.

- By default, before you do any multiplication, you have a coordinate system that doesn't exist. **It has to start off at the origin**
- Either rotate by alpha, or tilt the world.
- Tilting the world is calling a rotate function by itself.
- In WebGL, we almost always only work with change of coordinates.
- Just think of it how you would rotate a normal vector

Matrix Multiplication Review

- Use dot products for all 4 cells

Matrix Multiplication is NOT commutative.

- Rotate about the origin and translate it.
- Notice how it is scaling it in one direction and shearing.
- Rotate, scale, then shear.

If we change the order, it will shear first, then we are going to rotate, then we are going to scale.

Shearing: If you shear, think of it like a deck of cards being pushed aside.

- You don't have two eigenvectors. In a regular stretch, you have two eigenvectors and you can see they maintain the direction.
- The deck of cards shear does NOT have two eigenvectors; it only has one.
- This tells you there is a rotational component.
- Normal shear is for creating a tilted deck of cards.
- Shear occurs when you have non-uniform scaling.
- We can do a non-uniform that is NOT a stretch@
- A shear looks like a rotation in a different coordinate system.
- Does a normal parallelogram look like you applied a rotation at some way?
- If you rotate a square 45 degrees, it looks like a diamond!
- Weird case where you have scaled it on one way, so translations do NOT match in another way.
- You rarely want to shear everything in graphics; otherwise, you will get a stretched out person because you will get other parts of the people.
- **Rarely desired so we typically put away it at the very end.**
- Shearing basically means any non-uniform scaling.
- Actually a type of shear since it is a non-uniform scale + the rotation.
- If you don it in the wrong order, you will end up with a weird result!
- Make sure your matrix orders are correct!

Scale Matrices misbehaving

- Did you have a different image in mind?
- This is a whole mess so you have to very carefully execute the correct matrix order.
- World vs local
- You are scaling world coordinates so it is all his space in between!

Matrix Multiplication is NOT commutative

- Easier to think in terms of bases rather than points

Moving Objects

- Translation and rotation about z axis

Order of Transformations

- Hierarchy with a cube floating out in space

In both cases, we translate to form the green cube, then we translate and the rotating axis goes back.

- Graphics (174A) is a much lower-level introduction to understand what is happening under the hood.
- For game companies, you are using high-level functions where you don't have to worry about this
- JavaScript is easier than the C++
- C++ has way more files than the JavaScript
- Both are them are based on GL and what is creating all this is JavaScript

Stacks

- Very useful for graphics
- Take a snapshot of what your coordinate system looks like right now and then push it back on to the stack.
- We now have a snapshot of what A looks like, and then apply what you have popped to overwrite this whole thing.
- When you want the snapshot, pop to get it back.
- Used for undoing transformations

Snowman Example

- All we edited to change from the giant hierarchy is animation.js
- The only code we touched is at the bottom
- **GL cannot read strings easily**
- Series of three spheres
- This is a hierarchy: translated a certain amount from the bottom
- Do whatever is easiest to visualize for you
- Strategic to start in the middle because there are the most intersections
- We have various files:
- earth is just an image, so just upload our own image of how we want it to look like

In graphics, **y is up and down**

- Drawing the arms, if we don't want to do that, but rather, we want the cylinder to just be 0.7 m long.
- Use 4x4 to avoid gimblelocks?
-

Cylinders start parallel to the z-axis, so we need to rotate about the y-axis.

- Try all of these combinations!

Creating stacks

```
var stack = [];
```

model_transform = mult(model_transform, rotate(this.graphicsState.animation_time/20, 0, 1, 0));
this.m_sphere.draw(this.graphicsState, model_transform, blueThing);

W 2 Dis (Garett) 4-8-16
Examples:

- We can just use dx and dz to form a right angle

Homogeneous Matrices

- There is a 1 at the bottom because it is the difference between two points.
- The 1's are going to subtract out and you will just have 0
- When you add two points together, you get 1 + 1, which is 2 (doesn't make sense!)
- 1 is even more convenient because it lets us translate coordinate systems and when you do the transformation to a coordinate system, we don't want the vector affected by that and stay right where it is.
- Translating a vector should just move the vector and NOT change the direction of it.
- The zero at the bottom cancels out the whole last column where the translation is stored.
- The translation does nothing and the vector stays at the origin and is not affected by the translation

Matrix Multiplication Review

- Slide a second matrix above and below

Matrix Multiplication is NOT commutative

- Rotation is of the form 2x2 matrix (cos, -sin, sin, cos)

Shear: Angles get messed up; squares become no longer 90 degrees!

- Very last line of code is closest to the point in the matrix product.
- If we are looking from another perspective, let's look from a points perspective.
- All the points get spread out and translated.
- If you see any shape in your bee assignment getting sheared, it is because of non-uniform scale happening and you kept it there in the matrix products.

Translate our axes by 2, jump ahead by 4 instead of 2.
(0, 4) relative to the original space and draw it in those coordinate axes.

- Drawing this in a different set of stretched out axes results in a bigger face.
- Reading it right to left, interpret it as a moving coordinate bases.
- The evaluation of the matrix product doesn't matter because of associativity.

- Product is evaluated in order of the middle matrix and it will be the correct answer eventually.

Order of Transformations

- Your three core transformations are:
- rotation(angle, x, y, z)
- translation(x, y, z)
- scale(x, y, z)
- Follow a set of axes and draw shapes along the way

One of the parameters you pass in when you call draw and then you have lines that manipulate matrices

Stacks in JavaScript

- Making a stack in JavaScript is super easy
- A regular old array that can push and pop.

W 3 T Lec 4-12-16

- Looking at an object in 3D space and every object has a coordinate system, so it has an x, y, and z.
- How do you tell if you are looking towards -z or +z?
- It should start coming together that when you have an object in 3D space, the z is coming out forward in a right-handed system.
- This means you are looking at its -z value.
- Anything you see and visualize in the exam will make sense if you are asking questions.
- Find where the eye is in relation to the point on the 3D plane.

OpenGL Convention

- Coordinate systems are NOT always given, so we have to be in a position to find it.
- Gaze vector and we can use it to come up with a coordinate system.
- The other vector is g
- Imagine another vector coming up from the center of my eye and this is the up vector

Defining M_cam

- Start with a position for where my eye is and we also have a reference point.
- Forget the reference point for one minute, we are trying to build up a coordinate system for the camera.

Building M_cam Inverse

- Carry on with the inverse and in the next slide, we take the minuses there and then we need to find the inverse of the object as well.

Diana's way

- Nobody has given us a coordinate system, so we have to calculate it from scratch
- xyz-coordinates: Find out easily given a gaze vector
- This is still not sufficient for us to create a coordinate system
- We want to manipulate vectors to end up with another vector.
- This helps us towards building a coordinate vector.
- From two vectors, we have a gaze vector pointing forward and we can treat it as the w for the forward vector
- Any coordinate system is normalized:
- up-vector
- gaze vector
- w vector
- Take a cross project between t and g, and end up with another vector u (that represents the y value)
- Grab these two for w and u, and do a cross product to end up with a 3rd vector
- Brand new coordinate system describing this camera.
- If you did it one way, we just follow the right hand coordinate systems
- Same transformation as the one on the slides

M_cam and some line (a_i, b_i)

- Draw this line in camera space and it will end up drawing a line

Two vectors u and v

- Are they linearly dependent or independent?
- Look at if v and u are multiples of each other.
- Use this to describe a plane
- One of the slides can have linear combinations and this would describe adding vectors
- This green vector is a linear combination because it has a scalar x and a scalar y.

Summary of Modelview Transformations

- Camera transformation is a result of a change of basis
- You have to think about this outside of class

"objects are projected directly toward the eyes and they are drawn where they meet view plane in front of the eye"

- Depending on where you stick the camera, our entry point for the matrix is to first find out how many different ways we can project an object

Projections

- We can easily find what this point is because as we move the projector, we are dividing the distance by 3 or 2 and then we end up with that point.

- We end up with that specific position and this division will be cemented in our head.

A Basic Perspective Projection
- Going to be on all the exams!
- Make sure to understand this!

z = -N
- To my left is -z, this is what we started at this afternoon.
- The eye is at (0 0, 0) and we are looking at -z
- Here, we are saying that z looks at -N and the blue thing is a plane and N stands for near
- Sometimes we can have a near plane and a far plane.
- P' is creating a small triangle, and there is a bigger triangle where the actual P is.
- If we say that y'/N, this is equal to y/-z

In Homogeneous Matrix Form
- Represent them in homogeneous coordinates
- This gives us a general transformation matrix?

A Basic Orthographic Projection
- Get -N and everything else stays the same.
- Represent these three points on a projection plane and to do so, we need to preserve shape and size and multiply by this matrix.

Observations
- Foreshortening: this phenomena is called foreshortening and we don't start with using labels.
- Look for what we are doing and stick a label on it.

Notes: **Understand this lecture really well!**

W 3 Dis (Theresa) 4-15-16
Midterm
- Potentially end up in the discussion you are enrolled in.

Part I: Overview of Functionality
shapes.js, Shapes.cpp

Noteworthy Features
- Include files - just like C++, but found in index.html
- If you ever want to look at where files are included, you want the cube, sphere, and text to all be defined
- You want to use shape::draw() to do all the matrix multiplications and update everything to scale.
- Just a really quick pipeline of what happens.

- Look at the index.html file
- Look through everything in the index.html and all the matrices and vectors are here.
- Cannot use 3rd-party extensions, so you have to use OpenGL calls
- This is more of a math class so you understand the math behind graphics.

Matrix Review

- Test questions: you are going to have questions on matrix transformations
- Order of transformations is also fair game.
- Apply them in different orders
- Be able to draw transformations
- Do a couple of practice examples of drawing things in the same order.
- 10 multiple choice, 1 drawing question

Q. Will the test be OpenGL or actual math?
A. Probably actual math instead.

- Applying transformations applies to all points on the cube.
- If you cannot visualize it, pick a point on the cube and do a matrix multiplication
- Good way to check if you are off or not.

Transforms

- After you do the matrix, you have a different xy pixel coordinate that determines where the vertex shows up on the screen.

Q. Why are there two 0's at the bottom?
A. In any of the matrices, this is never anything other than 0.

- Homogeneous coordinates means we don't want this weird skew or shear in a direction that we never really use.
- Only use shear along the axes of the shape itself.

Transform Process

- Collapsing 3D onto 2D
- If we want to scale the cube up, use the model matrix for transformations
- Camera matrix can be treated as the basis
- We will see a different view for the camera
- Independent of what you have done for the cube.
- The camera is somewhere in 3D space, and it has some forward vector that is looking down.
- We don't want to project from the origin.
- Scaling a camera doesn't do anything
- It is just a coordinate basis.
- You don't want to skew it because you are mapping it to a monitor.
- In theory, you can skew it but everyone is working with a rectangular screen.

In the sample code for the snowman we can use “w”, “a”, “s”, “d” to fly the camera around.

- Go find that piece of code and apply a translation to move the camera around.
- Concentrate on HW #1 first.
- Focus on the static view, so we won’t be flying around
- If you don’t change anything about the template, you can still fly around and you should still do that.

Projection

- We have to map a 3D space into a 2D grid, which has x or y.
- There is no depth on a computer screen.
- Projection
- There is a 3rd projection and when you are modeling, you also model in 3D
- If you rotate a pen, part of the pen excludes itself.

Transform Process

- Projection matrix is used to map it so z coordinates to something on the xy-plane.

Viewport Matrix

- Whatever the thing you are viewing it through i.e computer or laptops screens
- We never actually see the viewport matrix.
- Applied for you at the end of the vertex shader.

Perspective Division

- Divide by its magnitude which is 1 for orthogonal matrix

Matrix Order

- The only thing that matters if you are going left or right.
- If you swap the order of two lines, one will come before the other.
- Order still happens during evaluation or in what you think in your head.

More rotation formalisms in 3D

- Quaternions
- High-level representation of 3D space
- Side effect of avoiding gimbal locks.
- Avoids curving path you have to take for rotating things around.

Gimbal lock with Euler angles

- High-level idea of Euler angles and quaternions
- Only rotate about the axes you have.

Powers of Matrices

- Our 45 degree matrix squared = a 90 degree rotation (try it by hand)

Screen Coordinates

- So far we only have considered
- World Coordinates == Screen Coordinates
- If the camera is up at 5, 7, 9, your screen coordinates have to be translated there as well.

Difference between Camera and Model Transform

- Left to right says this is applied to everything on the right-hand side.
- Going right to left, this is the point and here is the basis (change in coordinate systems) for our new view.
- Unlike a model transformation unlike going from right-to-left, you can think of a camera as sort of doing the inverse.
- Move coordinate system so that the camera is defined at (0, 10)
- That is what it does mathematically.

The Camera Transform

- For the camera, we are NOT being the object anymore, but rather we are being some image that the camera sees.
- Look for the most recent movement that the camera had.
- If you are thinking in terms of points, don't worry about this.
- Cameras are a result of the change in bases.
- Change of coordinate systems that defines your view.
- **Ignore this, think of it from right to left**
- The camera doesn't really matter because it is NOT a change in bases.
- In that case, the camera transformation is the same as any model transformation.
- The order we want to apply it in is C_3, C_2, C_1

Perspective transforms

- Parallel lines converge to a single point on the horizon line
- Let's say we have a camera with infinite resolution
- It will still find a point and we cannot break this 100 x 100 any further than that.
- If you have z, you will walk along the z-axis forever and it will run in parallel.
- There is no infinity in 2D, so essentially, when you have a projection matrix, it may change its depth.
- Depending on what projection matrix you use, it will start to look different.
- It is quite possible that the intersection is there and the angle you projected at was also different.
- You would get a different point of intersection!
- What happens in reality and what happens mathematically is quite different!

- I have erased the z-axis and mapped it onto a point in the x-y coordinates.
- One of the annoyingly hard to explain concepts.
- Projection onto a screen causes you to lose a dimension
- Let's say z is sticking out of a screen, the tip of your finger could be moving in x-y space and it will map to different points depending on what angle you are looking at.

Interpolation
- A way of having two things and you want to get two things in between
- Red and orange: you can interpolate all the colors in between

Barycentric Coordinates - easiest form of surface interpolation
- Smooth gradient from red to green and blue to red.
- One of the ways to dot his is through barycentric coordinates
- Each points would have some sort of weight on it.
- This affects how much interpolation is.
- If these had 3 weights, there would be a center of mass somewhere on that triangle.
- If it is heavier on another side, then the center of weight will be greater elsewhere.
- Center of mass should be in a triangle.
- Convex combination is one where all the points are greater than or equal to 0.
- Don't know the specifics but just how to interpolate colors

W 3 Dis 4-15-16
- If you learn the material, you will do great on the midterm.
- Pretty good scores if you show up next week.
- Not a lot of separation between you and the OpenGL calls
- Without a template, you just have yourself and an API call to the graphics card
- A bunch of function calls you can send to the graphics card
- If you look under the hood, you find the OpenGL calls
- In addition to the animation files, you have index.html which is the one you open to execute it.
- A couple of programs that we choose to package inside of it (shader programs)

Arrays for triangles, texture_coords, etc.
- Spaces that exist for the picture file rather than the 3D space.
- Big wrapper for all these numbers and these functions come up with all these numbers.
- Produce different arrays of #'s by the end.

function shape_from_file(name)

- Turn files into arrays to create additional shapes
- init_buffers: whenever we need a certain shape to be drawn, we can use one of these shapes cached in the graphics card.
- Just say “Hey, remember what a cube is made of? Draw one and use a certain model transform matrix”
- What each of these shape subclasses causes an array of points to be cached in the graphics card
- Overtime I make a cube object, it is calling init_buffers and setting aside space in the init graphics card
- Keep generating more and more points so it protrudes out.

The big axis thing in the demo gets one call to draw elements
- Has a whole lot of points on it.
- The more points in the shape, the more complicated the drawElements() call is
- For the cube, it is actually going to fork 36 times
- There are 3 forks of execution per triangle
- For the sphere, there will be thousands of vertex shaders called into action.

For the most part, you can post some nice custom stuff into the fragment shaders program
- You can have effects like cartoon effects, smoky effects, particle effects, etc.

Transform Process
- Always just one 4x4 matrix
- Same chain of pieces of a product
- Get you there in the same order
- Viewport matrix, projection matrix, camera matrix, model matrix
- By the end of this, there will be some z corresponding to the depth of the screen
- It will be x, y, z and w
- There will be a Perspective Division that happens to this final point
- This is NOT something the matrix could have done so that’s why the graphics card does it separately in hardware.

Matrix Order
- If you have one line of code at a time that pre-multiplies a matrix by another matrix, and then stores it, that is when it does start to make sense of talking about first and last.

More rotation formalisms in 3D
- In matrix form - you can combine these easily but you cannot just add them together
- If it is in a matrix form, you can just combine them into each other.

- What if you want to find the average between two rotations?
- Quaternions
- Some # in the space of l + ix + jy + kz
- Axis-angle is a vector in the familiar 3 directions

W 4 T Lec 4-19-16

- Idea for Homework 2: Super Hamster Ball
- Modeled of Super Monkey Ball

Clipping

- If we clip without going to a canonical view, it will be different from clipping from an canonical view
- We are trying to get these lines from perspective and make them parallel

Perspective Projection

- For us to convert lines to parallel, we push out the origin and the viewpoint to get out into infinity.
- We do two things, push the viewpoint out to infinity and we calculate this by dividing by the homogeneous values.
- We are building out this generalized matrix to get a canonical view
- We also need to go to the next step, which is to push out the viewpoint to infinity and we need to divide by the homogeneous value.

Projection Matrix

- These 3 matrices are all multiplied out and we end up with a generalized, canonical view matrix as defined by OpenGL.
- Normalized for everything but the z-coordinates
- The z is going to easily become our guide to when we want to draw things out in screenspace.
- We will take into account the z-stacking of objects and see which one is the closest to the projection plane and render that.
- A couple of algorithms like the z-buffer algorithm to very well-known algorithms that do this.

z in NDCS vs -*z* in VCS

- Look at z values for the viewport and this is the relationship between both.
- z-values don't translate and they aren't translating accurately.

Screen coordinates (2D)

- Mapping to screen space
- We know that we index the pixels by the column *i* and row *j* with respect to the origin.
- This pixel is right at the center, which means we add a 0.5 to i and j instead of just i, j

- Take this into account when creating the generalized matrix to go to viewport and screenspace.

Viewport Matrix

- Multiplied by 1/2 to account for the 0.5 in the pixels

GL Functions in the Pipeline

Things may start changing i.e. performance costs when we are dealing with stereo

- Whenever we are trying to display something using an oculus or head-mounted display, the image is duplicated
- If we duplicate across, we render separately for each eye.
- We need to be able to assign tasks to hardware and software
- As we improved GPU's, it made sense we wanted more control over the rendering process.
- Why do we care about this?
- We start by playing around with 3D objects in a virtual world and we want to play them in a 2D screen.
- We have to undergo a series of six transformations every time and apply them to our objects that we are trying to rotate and play around with so we can display them accurately.
- Steve Jobs - trust that what you are doing right now will be relevant in a few year's time.
- We don't know how graphics could come in handy when we are 30 or 40 years old.

A Line Rendering Algorithm

- A bunch of pseudocode and we should understand this!
- We start with the first matrix in mod space and we can find the inverse cam to translate the view back to the origin.
- Much easier to make calculations rather than trying to consolidate
- The viewing camera has one basis and the world has its own coordinate systems.
- Make a generalized matrix and a prospective one, and we can then multiply by a viewport matrix.

3D Clipping

- Implicit equation of a place and solve the for the other.

Bases

- A 2D vector can be expressed a combination of any pair of non-parallel vectors

Normal vectors

- Vectors that are perpendicular to another vector
- Using implicit equations of a line
- Parameter equation for the line from P_a to _P)b

Z-Buffer Graphics Pipeline

- Canonical transformation when we convert the graphics pipeline
- Do some manual clipping
- Reused implicit equations to find this and perform this clipping
- Generally, we need to clip as part of the geometric transformation that we need to do.
- Take it to the viewport for another transformation
- Implicit equations help us for geometric transformations
- There is quite a bit of use for both equations
- Shouldn't be a term that you look at and what is it.

W 4 R Lec 4-21-16

- Draw the line with 5 subroles
- Straight lines should appear straight
- The line should start and end accurately
- Match the end points with the connecting lines
- Lines should have constant brightness
- Lines should be drawn as rapidly as possible
- Augment it to get to the point that it is the fastest.
- This is where we understand ahead of times things we can draw.

Problems of drawing a line?

- How do we decide which pixels to illuminate?

Equation of a line
$y = mx + b$
start at (0, 1); m = 3/5 b = 1
Draw this on paper

- Calculate the next value -> the delta value
- What is the difference to the next point?
- Deal with differences between each point
- Get rid of multiplication and figure out a # of values

W 4 Dis (Theresa) 4-22-16

- 10 MC questions + Drawing question
- Shear: Non-uniform scaling

Pumpkin Drawing Example

Review linear algebra!

W 4 Dis (Garett) 4-22-16
Click & Drag Camera

- JavaScript closure

- Everything inside the curly brace has a callback for the event listener.
- This expression is going to make a function object in JavaScript that is going to get called
- If we do that, we are inside an inner function and it is in a function that is in a function
- You lose the “this” variable and it is going to crash
- JavaScript is meant to have functions that are one function deep.

Part II: Interpolation

- Linear combination

w = a_1 * v_1 + … a_m v_m, a_1, …, a_m in R

- Affine combination
- Preserves affine combination relationships
- You are interpolating a new point
- Affine preserves betweenness and any points between them will end up between them after the transform.
- Looser than linear transforms, while linear transforms are more restrictive.
- Take us outside the convex hole of the shape if we take the outer boundary and wrap it around the shape.

Barycentric Coordinates

- Hang three weights off the triangle and this forces you to put your finger at a certain point when you try to balance it.
- The weights will determine where the point is where your finger has to go.
- These weights are pulling the finger towards each point and you end up by altering the weight to move P to the extreme points.
- You could have colors and have the weights represent those.
- The more of this point you use, its weight is going to pull P towards the red side of the spectrum.
- We can have values like color or a normal vector and smear it across the triangle.
- easiest form of surface interpolation

The Process

- Starts somewhere in world space and has to project onto an image on the screen
- The vertex shader applies the matrices to each of these vertices and they need to land somewhere else.

Q. When it is at the vertex point, we are transforming the object to create the world space like this?
A. We will use very simple #’s and some unit vector and the model transform will get applied here and this sends them into world space.

- The projection transform will stretch things out and this will be big enough to fit on your window.
- Create your model transform and put it into OpenGL.

- More parallel since this thing will run on every single point, and we need to do this sequentially, while the graphics card can run this in parallel.
- Much more efficient

Varying Variables

- Automatically smear themselves out based on which pixel we are trying to cover.
- We are currently iterating through those pixels and our normal is gradually changing as intermediate values for those extreme points.
- Interpolation is built-in as a major variable type in your shader.
- There will be variables that are NOT associated with every single point and instead of having a color at every vertex, we will have one model transform for every other vertex.

Uniform Variables

- Affects every variable uniformly
- Are we going to texture this object or not?
- Doesn't change from point to point.

Part III: Making a trivial fragment shader

- Interpolation will probably be a midterm question

Interpolation of "varyings"

- vColor that lives at every vertex and we are drawing triangles at every vertex.
- We need the colors at those extremes, but it shows up as smeared across when we run it.
- Computes intermediate values rather than using the extreme points.
- Helps us to make the scene a little bit more realistic.
- We can draw things that are smaller than the triangle

Part IV: Drawing practice for midterm

- Post-multiply example

Be careful; it can be tricky!

- The 11th one has subparts to it.

LookAt(): Easy way of looking at the camera so during an animation, the camera will be moving around.

A lot of projections on the midterm!

- No gimblelocks on the midterm

What happens to parallel lines during the transforms?

- Pierce into the image plane and a prospective transform warps everything and lines up into the image plane.

- The wood pieces of the track are parallel to the image plane and any phenomenon does NOT get affected by the image transform
- Some lines stay parallel, while others do not.
- We are much farther down that line and that is where the track is.
- Ratios along straight lines get ruined by a perspective transform.

W 5 T Lec 4-26-16

Game works

- Concurrently draw holes and scan it out, which is much more efficient
- Command we will call in this API and there will be a procedure after the other and we need to combine things so we do a double for loop
- This whole API is to alter the graphics pipeline and make it more efficient
- See how we can use it and make it relevant to you.
- Check whether there is enough popularity to create a game engine for VR.
- We needed at least 30 people interested in taking a class like this (CS 188)

Multi-Resolution Shading

- Given knowledge of the graphics pipeline and the viewport, this document should be used to scale viewports and show viewports here.
- If you read through it and you get it, you have done well!
- Midterm is feedback to tell you how well you are doing.

Polygon

- Collection of points connected with lines
- How do we connect these vertices to the graphics pipeline?
- We only have one messy example of a data structure!

Triangles

- What is a problem in taking these vertices and pumping them through a graphics pipeline?
- The solution is to organize these vertices in a way that they are NOT duplicating.
- When we describe the triangle, we don't want to duplicate the vertex.

Testing Concave vs Convex

- What is the simplest test?
- Concave > 180
- Convex <= 180

Triangles

- The most common primitive

Plane Equation

- Normal/point form

- Ax + By + C = -N * P + D [Normal * P + D]
- This will be in the final!

Polygon Attributes

- We want to do revision on parametric equations
- Next three weeks
- Go back to parametric equation and do supplementary material exercise
- On Thursday, there is NO lecture!
- Work on assignment 2
- Once we come back, we are going over illumination and we want

Assignment 2 out of the way.

2D parametric curves

- A parametric curve is controlled by a single parameter and has the form:
- **p** = (x y) = (g(t) h(t))azZZ

W 5 R O.H. (Garett)
Topics to Cover

- Line equation forms
- Interpolation
- Geometry
- Projections (4 questions)
- Pumpkin example (2 questions)

See if you can do the pumpkin example with both a basis perspective and a points perspective

- Get two out of three methods to match

0. Matrix products and then doing multiplications out

Perspective division
P * C * M * P (affects the last P term)

- Responsible for converging in the distance.
- Go over how to draw shapes in this class because it comes with a

capability of dumping contents into shapes

W 5 R Lec 4-28-16

- Read slides and understand it, no need to read a textbook!

5 Questions on Transformations
4 Questions on Viewing Transformations

- Transforming paper using OpenGL and drawing something
- Simple primitive that should be okay.
- A little tricky so brush up on rotating 90 degrees left or right
- Understand Affine Combinations and what the mean as well
- Reflections -> understand these and how to translate it.

- Difference between implicit and explicit equations
- Be able to understand and derive the *i* at the origin and viewing transformations

- Understand if they keep the ratio or not
- Properties of a triangle
- **DO NOT STUDY MIDPOINT ALGORITHMS**
- Understand viewing transformations and transformation algorithms well
- OpenGL will be a basically a translate and rotate thing
- Best practice is to know where to start reading the OpenGL
- Translate it multiple times

Triangles: barycentric coordinates

- Understand your parametric equations to memory
- p_0 + t_1 (p_A - p_B)
- Have two vectors (c-a) and (b-a)
- In order to advance these points along, we would multiply them by their respective scalars.
- Issue is that this coordinate system doesn't show us the location.
- All we need is for this triangle to map out where A is
- Express points at different basis and we can establish vectors as existing
- We need to know where its exact location is and we need to adda point to it.

Ray/line-triangle intersections

- Simple properties (if the two scalars == 0, we will end up with a value of vector *a*)
- If we multiply one vector by 0, you get 0
- If we plug in values, we can get values for other vectors.
- Don't need to memorize the; just know how to calculate these.
- We can calculate the different vectors
- Made of parametric equations and then we can find exact values.

Interpolating: finding values and it depends on the in-between values

- Map positional values to correct values in textual space
- Values will always be between 0 and 1 for S and 0 and 1 for T.

h_r == w_r

- This means we introduced a projector at some distance away.
- We have dealt with this problem before; if we are transforming an introducing a projector, then this isn't an accurate presentation of these points.
- If we move projector closer or farther way, r is going to be different.
- To solve this, we homogeneous by dividing by w or h_r
- This gives us 1
- If we are going to interpolate, we need values in between 0 and 1

Translation of a line proof

- We have added a transformation matrix to this parametric line.
- We want to interpolate on to the texture values so we can get the right numbers and mapping.
- If we apply a transformation to it, we will get the same values in world space as we would in homogeneous space.
- To solve the problem, we need to move away from screen space and make sure we are dealing with values NOT from screen space.
- Make sure our values are in homogeneous space.
- This is the main reason we are doing this.
- Brute force manipulation of an equation
- Projector forms a line with the screen and we need to describe it in parametric form.
- Change the letters to w

- This screenspace to texture space is problematic due to foreshortening, so we need to use the T_value and we are going to get the wrong values.
- General formula for barycentric triangle and apply this to u and v
- All this is saying is use a parametric vs barycentric equation
- Barycentric describes its location and we would use the same equations
- We need 2 vectors for this.

Reminder: In-Between Points

- Using “lerp” for interpolation
- Removes you from the equation and we are going to plug in a, b, c
- We are doing this to prove that transforming doesn’t give us the same value.
- Interpolating in screen space gives different values than interpolating in world’s space.

- We need to intersect it with an actual scan line of screen space.
- When we intersect, we find where the scan line (in my case u) intersect with the triangle and we can find that point.
- Interpolate and find points in that line.

- Using (s, t), we can color this teapot and use a sine function and feed that texture coordinate and it will just loop it.

Pseudocode algorithm

- w is the units, how far the projector is from the pane
- f(t) is world beta or world gamma
- Put real numbers and for all the examples, we are talking about a point in space and near we are going to project things and those 10 units could be described as a parametric equation too.
- 7.8 is just trying to extract the t value we want. Use brute force to manipulate the equation

To avoid foreshortening, we need a correct way to access values in this triangle.

- We need to be able to map things to the correct textures we want.

Light Maps

- What do we do?
- The solution is to create gray textures with different intensities
- We could start achieving stylistic effects if that's what we want to do.
- In any of the gaming classes i.e. 3D-real time animation, they have to light their scene properly.
- Why can't I manipulate my lights for this scene?
- The textures will be baked in within the lights.

CrazyBump Example Video

- Displacement map changes the shape of the object and we don't want to go into these maps quite yet.
- He is playing with the normal map and it saves the values of the normal and when we move the light around, it is interacting differently and gives you that detail.

Create a displacement map and we should take into account normals from various positions

- Otherwise, your objects will NOT look detailed enough.
- bitmapping is very important
- Different people and different students out of Utah University's collective knowledge base

Next Week:

- Tuesday: Midterm
- Thursday: No class

W 5 Dis (Garett) 4-29-16

- There is a tricky part to the midterm and it should be possible to get 100% if you are careful.

Scene timing idea - from Piazza

- Keep multiple copies so as time keeps ticking, we will eventually call display() and the time will have exceeded the other ones.
- This means we will be within that window of time to execute some scene.
- We want it to last 7 seconds and keep in mind everything is in milliseconds.
- Turn it negative manually so down here that means we are NOT in that scene.
- Use conditionals (if statements) to have the program do different things at different times

Advanced Project 2 Topics

- If you have a plan to make a scene about the beach, you are busy making assets for it and drawing a tree or a boat.
- Drawing the boat, drawing the water, etc.
- You need to make it nice to have a technically advanced goal in mind to drive you towards finishing your animation.
- You can have a tree data structure and at each branch, there is a model transform stored.
- Take the first branch's matrix and multiply it by its second matrix.

- Ask Garett how he made his game!
- Copy and paste commands to make more nodes of the scene graph
- There is a matrix at each branch of the scene graph and it is like an n-ary tree with each parent affecting its children because the matrices you multiply each other.
- You end up with a new matrix at the bottom.
- Each shape you see is drawn using one of the nodes of the tree.
- **The balls are from a 3rd party library.**
- **It was for the physics engine**
- **Bullet physics engine in C++, 3.JS has some physics stuff but if you use that, you cannot do the drawing stuff.**
- **You need to extract out physics part of 3.JS**
- **Cannot replace the draw stuff, but you can use physics**

- Shadows - Use shadow mapping because it is one of the easier ways to do it.
- Shader programs - Pretty flexible already and we have something like fong (?) shading.

hermitMatrix

- Makes a curve between two points and does so by repeatedly drawing objects along the way of the curve.

Quaternions

- Done for advanced camera tricks along a sphere
- Uses a geo-designated shape.

Draw assets first

- Tell a story about the objects (assets) you manage to make

Part I: Specifying your own shapes from scratch with polygons

- Don't use the cube, sphere, or strip
- Instead, you can use your own strip out of points

W 6 Dis (Garett) 5-6-16

- Final is all multiple choice and 50 questions

lookAt Example
this.graphicsState.camera_transform = lookAt(vec3(0,0,0), vec3(0,0,-1), vec3(0, 1, 0));

this.graphicsState.camera_transform = lookAt(vec3(0,0, 0), vec3(0, 0, 1), vec3(0, 1, 0));

- Use lookAt to change your camera angle and put an animation

Look at mv.js to see everything that can happen within a matrix

Project 2 Requirements

- Hard edges and shading and it is because the normals on each triangle is pointing the same way
- They are NOT fighting over the same vertices and it has separate triangles for every single face.
- Cherries are just spheres and they made vertex arrays for it
- Flat shaded object
- Fong shaded object (default)

Space Bomber

- Big barrier between getting 100% and placing in the Top 5.
- You get a lot of extra credit under Diana
- Feasible to get 100% without working too hard on the final

Requirements

- lookAt - Put it in the project
- Design your own polygonal object
- Show at least one flat-shaded seam
- Texture something
- Real-time speed (not hard)
- Display the frame rate on the graphics window
- Casually investigate how the other strings on animation.js and put this in
- this.animation_delta_time = time - prev_time;
- frames per second: 1 / this.animation_delta_time

Flat-shaded: Approach the edge from one color

- You make it so that your triangles are NOT fighting over a vertex
- All the normals go in the same direction
- Normal belongs to a triangle rather than a particular point

Curves

- Interpolate down to a single point, which lets you turn several points into a nice, smooth curve.
- Plug in each line equation and it should end up with a big polynomial and your current points being based on all those different terms
- t is scattered out all over this polynomial
- You want a curve that goes through the points

Motion

- Treat the like they are just boxes and run a test to see if they are overlapping

Collision Detection

- Model everything like a sphere and the rotation needs to be possible
- Make it the model transform's job to scale it out.
- Check if it is within 1 (if it is, it is inside the sphere)
- Take two stretched out spheres and take one of them and undo it back to the origin

W 8 T Lec 5-17-16

Curves

- Sees how Pixar applies curves
- If you take a cone and intersect it with a plane, you get a conic section
- You have the first two scenarios and in 2D, the intersection of two lines yields a point.
- In 3D, the intersection of two planar surfaces gets you a line

2nd degree implicit representation

- All possible curves show these coefficients (look in notebook)

Conic sections - parabola

- Different types of curves and their mathematical representations
- If you look at the parabola, it has a focus point and a directrix, which is a line that is equal distance from this focus point
- It is a good place to think of curves.

Present Demitri's slides and figure out all this crap with TheKevinWei

- If it is one slide, it is probably more work.

Parametric equations are curves based on the coordinates of the curve.

- Parameter through time is tracked through a position.
- Track things using parametric equation and calculate the x-y coordinate.
- For a straight line in 2D, if we know the coordinates, we can express it in terms of given points.

2D Curves: ParametricForm

- An ellipse of half-width w and half-height h centered at 0

Bernstein Polynomials of Degree L

- L + 1 control points

W 8 R Lec 5-19-16

Homework 3

- Tiny matrix library like the JavaScript one we are used to.
- C++ project, not JavaScript
- Three code files and we need either a Visual Studio or Xcode project
- Maintain an array of spheres and lights
- Comment your code properly

Cubic Space Curves

- All functions of cubic polynomials
- Square polynomials are complicated enough
- Alternatively, you can condense this into a matrix form for the 3rd entry.

Rewriting with Geometric Constraints

- We want these points to be multiplied by the basis matrix and we can go to what the coefficients should be.

Equation for the Hermite Curve

- Here are the 3 matrices right now and that second equation makes it easier to see the constraint functions
- Look at it horizontally
- Blending functions are like scalars for what these points are
- These are tangent to the points and they correspond to the matrix.

Hermite Blending Polynomials

- Linear combination of these four
- Last two functions blend the two tangent vectors
- Hermite polynomials are used in graphics
- In practice, they are good for fitting it in an existing surface and you want to see how changes in the tangent vector affect these.
- Easy to piece together these termite curves and each piece of specified by a join curve.

Matrix Form for Cubic Bezier Curve

- The point we will draw at is in the middle, and the line will curve along.
- First, you write an equation to do this and do it one more time.

Converting Between Cubic Spline Types

- Use simple linear algebra

Catmull-Rom Splines

- Also, we will have tangents for these curves and please ask for just the scalar of these tangent lines
- We would NOT choose to be greater than one

B-Splines

- Knots are NOT on the control points
- Rather, they are inside the control points

- Each know is C_2 so it is continuous up to the second derivative
- If you interpolate between the points, when you are doing this, you aren't going to use interpolation.

B-Spline Basis Functions

- Each of these blending functions are polynomial functions
- The bottom part says there is a convex hole property
- Control point determines where the hole is

Wish List for Blending Functions

- Blending function should be 0 and that control point shouldn't affect the shape of the curve.

W 8 Dis 5-20-16

Q. Can we still see our midterms during office hours?
A. Garett will have all of them.

- Assignment 2 is going to be graded and we will see.
- They are all downloaded and we can start grading them
- These should be back to you next week as well.
- If we do hold the contest, it will be in discussion.
- Pretty big undertaking to hold it.

Raytracing

- Non-computer graphic result: calculating from rays that origin from a source of light and see if it hits an object to determine how they illuminate to see how a scene is completely lit.
- Computer science graphics definition: Instead of calculating it from the light source to the eye, you calculate it from the eye to the light source
- N x N pixels and you calculate pixel by pixel and you trace the ray back from your eye and cast it to the viewing scene.
- From that object, you calculate the color and the brightness of that object by calculating what light sources we have and what fraction/ambinent light.

Instructions

- Only need to render spheres
- Camera is situated at origin (0, 0, 0)
- Right-handed - looking at negative z-axis

Raytracer Project

- Three file template
- No code learning or OpenGL calls
- vectors, for loops, input/output files
- Cons
- Not real time
- Do all the calculations, render it, then output it.

- Parsing the file correctly and if you get those numbers wrong, you get the image wrong.
- Raw binary and copying a file
- You can use OpenGL for debugging but it is impractical for this template

Results
- We want to see the actual reflection's lighting correctly.
- Extra step: Culling objects inside the near plane

Grading Scheme
- Gimp/Photoshop
- specular, diffuse, shadow, reflection

Phong-Blinn Shading model
- Specular and you will still get slightly different results from (R dot V)^n versus halfway vector (H dot N)^n

Phong Shading model
- For every single pixel, it will return a color that is rendered on the screen
- K is just a coefficient and you want to solve for illumination of the object and if you get this part correct, you should see flat colored spheres against your background
- In addition to that, for every light source, you should calculate if they hit the sphere and then you want to check if any of the other spheres cast a reflection on your current sphere.

File Parsing
- Your only job is to set it up so that we make a bunch of if-else statements

getDir()
- This pixel maps somewhere onto the world coordinate view screen
- How do we know which point this pixel maps to?
- You have to start here to cast a ray into the world.
- This is the first part of the assignment and you have to give a pretty big headstart.
- Something that maps one to one in pixel space and this has all the points and the only thing you are missing is the z-axis.

Q. What does this function do besides mapping your pixels to your world space?
A. Return a vec4 that is a direction to the array
- Build an array from the camera to the world.
- Try to render pixel (0, 0)
- If you want to render this pixel, we need a ray to trace and see what it hits in the world.

From Lecture slides
- Ray-Object Intersections

- Only takes one value and it returns either a true or false.
- Passes a lot of things by reference
- Check if you intersect with a sphere or not

Most useful lecture slides
Summary: Raytracing
- Pseudocode for the project

Phong
- ambient
- diffuse
- specular
- shadow

W 9 T Lec 5-24-16
- Contest will be held sometime soon

Height Field
- Geographic elevation map that is usually in black and white where black is the lowest elevation and the white is the highest elevation.
- The purpose of these height fields are primarily terrain
- Mountains
- Valleys
- Less complicated than calculating things one by one for ever single vertex

Example Height Fields
- Put it into the sin equation, and it will repeat your pattern over and over again

Computing Surface Normals
- We parametrize in two variables here

Quadric Surfaces
- Used as primitive shapes to construct other quadric surfaces
Sphere is a general form of an ellipsoid
- Used most often in this class

W 9 R Lec 5-26-16
- This viewing transform isn't one that projects it onto a plane
- Now you are in 2D instead of 3D.
- This is after the camera has been placed in the world
- Is it shaped like a perspective or orthographic?
- Perspective division happens which is a non-linear operation because a matrix cannot do it, so all this chain is a chain of matrices multiplied together.

Viewport transformation

- Stretches out to match that window size and then it draws it.

Backface Culling in the VCS

- Polygons that are facing the positive z-axis or going to be facing you
- In orthographic coordinates, we just compare to the z-axis.
- This just means that it is following one orientation on the view plane depending on the side we are at.
- Here is taking these polygons and extending them so these planes go on forever.
- We shouldn't see this polygon

- Pick any three of the points and turn them into a pair of vectors

- If you compute this for the plane, and you check the sign for this.
- Take the camera's x, y, and z. If you are looking dead-on to the plane, it will usually be positive or negative.

Backfire Culling in the NDCS

- This point is falling on this pixel but it isn't quite there yet.
- In that coordinate system, you can use the z-coordinates and determine this.
- Trick used to save time.
- You will see shapes that maybe faster to do this to eliminate triangles first.
- Otherwise, you have to use CPU time and you want to try to determine which triangles matter and which don't.
- Usually, it is fast enough to keep up to calculate things in parallel and process a lot of things at once.
- It is debatable when you want to use this or not.

Visibility Algorithms

- They find out if you are looking at the backside of other things in some other ways.
- Raytracer - assuming your intersect function works and you are correctly recovering the color and it is the closest one to you, you don't have to do backfire culling because you did the comparisons to every sphere already
- In this case, you never have to draw the backside of the sphere
- Make a 3D grid that takes up all of space and just keeps going off to infinity.

- Only a few thousand intersections and you have only computed one ray, so you have a million rays you have to do.
- You can only process the pyramids that matter, and this is kind of a state of the art way to do it.

- Ray tracing goes in the opposite direction
- Z-buffer is what we used in project 1 and project 2

Project 3 Nots

- vec4 trace(ray)
- (r, g, b, 1)
- We gave you a version that works and the color is black no matter what the array is
- Default: return vec(0, 0, 0, 1);
- You can test this function without finishing this whole thing
- The intermediate steps might want to test values like are my ray directions correct?
- Just return those as a color and we don't want to return our final answer with all of them together.
- Test and see if it is working before going on to build everything else.

0. Replace with background color

for every sphere
{
    t1, t2 = Quadratic solve
}

W 9 Dis 5-27-16
Project 3 Notes

- Figure out how to store information using Phong Shading model
- Store them in vectors
- Inverse matrix can be helpful
- Fill in getDir which we went over last week regarding generating the array
- You want your retracer to run in reasonable time.
- Intersect function
- If you get this done, you will be 90% done
- Plenty of ways to do it.
- Have a sphere at the origin, subtract 1 and get 0
- Take this ray and at some t, we are going to have it intersecting the sphere and it will equal 0
- You want to use parametric form because it is easier
- If we intersect the sphere, you can get it to |S + tc| and set it to 0
- If you want to find the intersection point, what is the first step we want to take?
- Calculate S