Computer Graphics

COMP3421/9415
2021 Term 3 Lecture 4

What did we cover last lecture?

Starting to look at 2D Rendering

- The OpenGL Pipeline
- How pixels are coloured in a polygon
- Textures

What are we covering today

Making games in 2D (not quite, but close!)

- Textures and some subtleties
- Transformation Matrices
- A breakdown of a Sprite based game

Textures

Textures Recap

What we've seen about Textures

- Use of image files in polygon rendering
- Using Vertex attributes to "map" vertices to coordinates in the image
- Using Fragment shaders to pick up colours from the image
    - and interpolate them across the shape

Images credit: id Software

Why are Textures useful?

Textures vs Coloured Vertices

- If we colour verts . . .
  - We'd need a lot of verts to do detailed colour patterns
  - Simple, flat surfaces like walls, carpet etc would need a LOT of verts to represent things like weave patterns or wallpaper
  - All these extra verts will take a lot of time for our GPU to process and a lot more memory to store
- If we use a texture . . .
  - We can use simple geometry and allow the texture to carry the details
  - Minimal amount of verts
  - Minimal amount of extra work in the vert and frag shaders

Simple Geometry, Complex Colours

Image credit: Gearbox Software

Texturing in the OpenGL Pipeline

We're starting with using 2D Textures

- We load our image into OpenGL (code details in your tutorials)
- We sample from textures using a coordinate system from 0 to 1 (floats)
- We can add texture coordinates to our verts
- This allows the fragment shader to sample from the colours in the texture

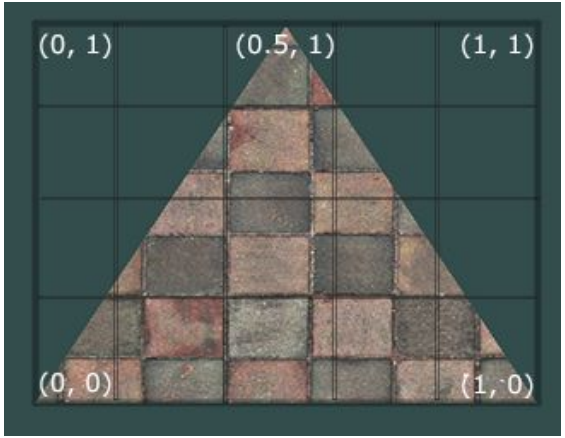


Image credit: learnopengl.com

Texture Wrapping

How do textures deal with sampling outside 0-1

- We can sample past 0 and 1 if we want
- Sampling behaviour changes depending on how we want to deal with this
- Default just repeats the texture again
- But other options have their uses . . .

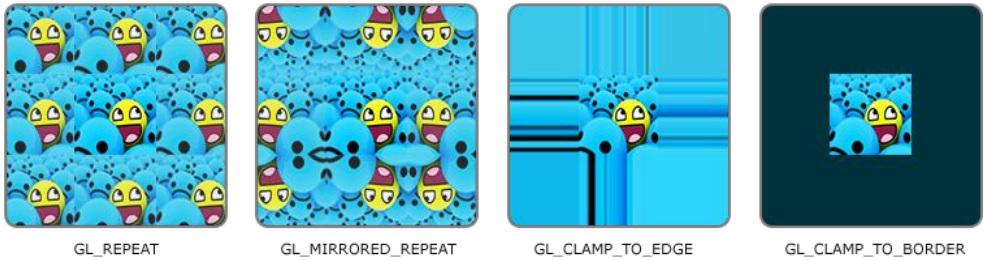


Image credit: learnopengl.com

Texture Wrapping Options

Why might we go outside the 0-1 range for texture sampling?

- Repeat: Wallpaper, wire mesh or other constructed surface
- Mirrored Repeat: Grass, dirt or other natural surface (google seamless grass texture)
- Clamp to Edge: hmm . . . you are only going outside a little and don't want things to reappear?
- Clamp to Border: Posters, stickers, decals? Treat the texture as a one off that never repeats

Texture Filtering

It's not a one to one match between fragments and Texture Pixels

- We call the Texture Pixels Texels (they're definitely not pixels!!!)
- When a fragment samples, it's not guaranteed to land in the middle of a texel
- OpenGL has different options for this:
  - GL Nearest - take the texel that you're the closest to the centre of
  - GL Linear - Take a linear interpolation of the colours in all the texels you're near

Image credit: learnopengl.com

Mipmaps

Do both of those filtering options look bad?

- The ideal is 1 fragment samples 1 texel
- Textures need to be sized based on the object?
- This is awkward if we can move around and objects change size based on our distance to them!
- Mipmaps are sets of textures that all represent the same texture at different sizes

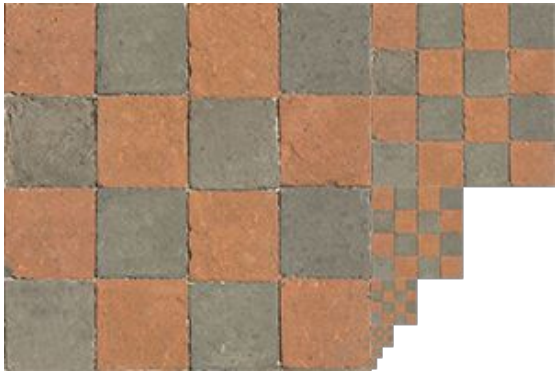
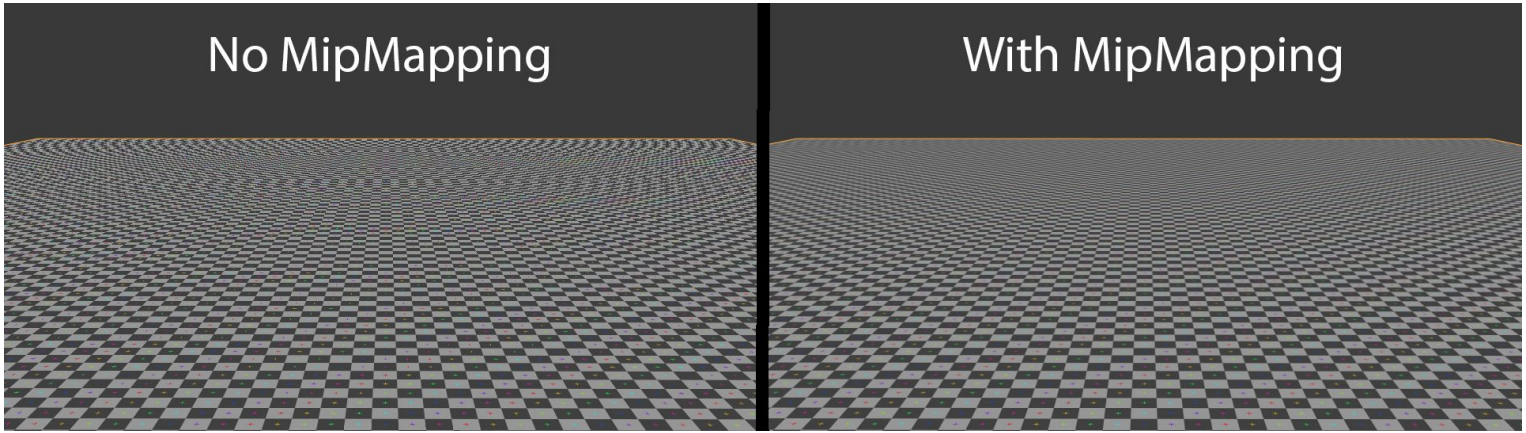


Image credit: learnopengl.com

Mipmaps



- Sampling without mipmapping can lead to some strange patterns
- Mipmapping allows textures to degrade gracefully into the distance

Matrix Transforms

Transforming Objects

Our vertices have been set in stone up to this point

- Wouldn't it be interesting if they were more than meets the eye?
- We should roll out a new technique to change the position of vertices
- Option 1: We do this manually
  - Write new vertex positions and rebuild the VBO 60 times a second
  - While technically possible, this is very cumbersome
- Option 2: We use the Matrix ~~of leadership~~
  - Linear Algebra gives us some easy tools for transforming vectors
  - 2D or 3D vertices happen to be very similar to maths vectors

Image credit: Hasbro

Vectors and Matrices

How well do you remember your Linear Algebra?

- We're not going to go over it in lectures
- But you might need to refresh:
  - Vector arithmetic, especially dot product, cross product and normalisation
  - Matrix arithmetic, especially multiplying matrices and multiplying matrices and vectors
- Vectors are directions measured by coordinates
- Vertices can be thought of as vectors starting at (0,0) and ending where the vert is

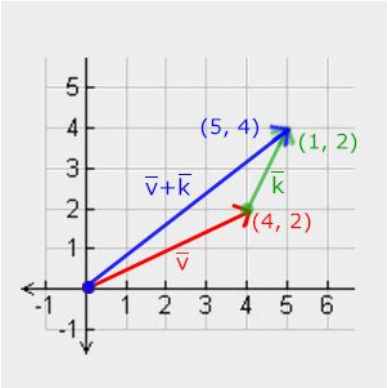
Vector Math in a visual sense

If we're going to use vectors in a visual system . . .

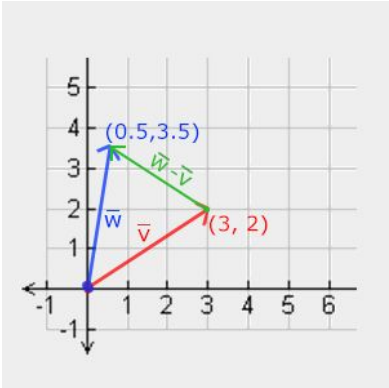
- Adding vectors is like following them on a journey, each vector one after another
- Subtracting vectors tells you how far apart they are
- Dot product gives us an idea of whether two vectors are aiming in similar directions (great for lighting and reflections)
- Cross product takes two vectors and gives us another that's perpendicular (90 degrees) to the other two (great for building up coordinate axes)

Visual Vector Arithmetic

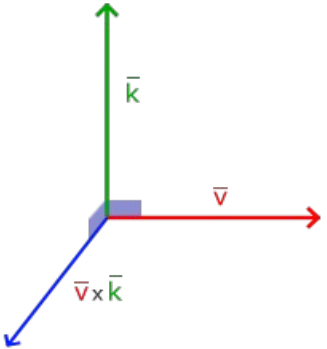
Adding



Subtracting



Cross Product



Images credit: learnopengl.com

Applying Matrices to Vertices (by multiplying)

We can multiply a vector by a matrix

- Which means we can apply a matrix to a vertex
- The output of multiplying a matrix with a vector is a vector
- So what it will do is possibly change the values in the vertex, which changes its position
- Even more interesting is applying the same matrix to all the verts in a shape or object
- We have some pre-made transform matrices that we'll use a LOT in graphics . . .

Vectors in OpenGL

x,y,z,w

- There's always one more coordinate than the number of dimensions
- We call this 'w'
- For the moment it's just going to be 1
- It won't make a difference now, but definitely will in the future
- So a 2D vector (or a vertex being transformed) is: {x,y,w}
- and 3D is: {x,y,z,w}

Scale

Changing the size of an object

- This matrix can change how far vertices are from the origin by a multiplicative amount
- If applied to multiple verts in an object, they will change the size of the object

Scale x	0	0
0	Scale y	0
0	0	1

Translate

Moving an object

- This matrix can move points by a fixed amount
- Applying this to all the vertices in an object will move the object to a new location without changing the object

1	0	Tx
0	1	Ty
0	0	1

Rotate

Spinning an object

- This matrix will rotate a vertex around (0,0)
- If applied to the verts in an object, it will rotate the entire object around (0,0) without changing the object

cosθ	-sinθ	0
sinθ	cosθ	0
0	0	1

Combining Transforms

Matrices can be multiplied with each other

- This combines the effects of the two transforms
- Remember that this is NOT commutative
- A.B ≠ B.A
- The order you use transforms is important!
- There is no limit to the number of transforms that can be combined into a single matrix

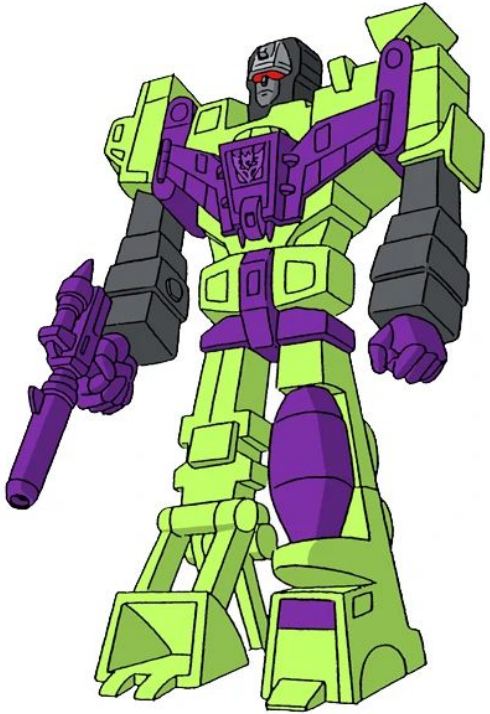
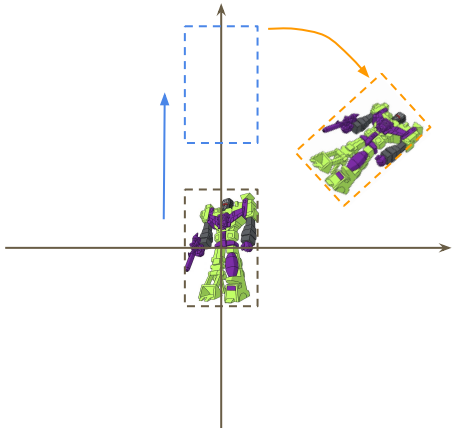


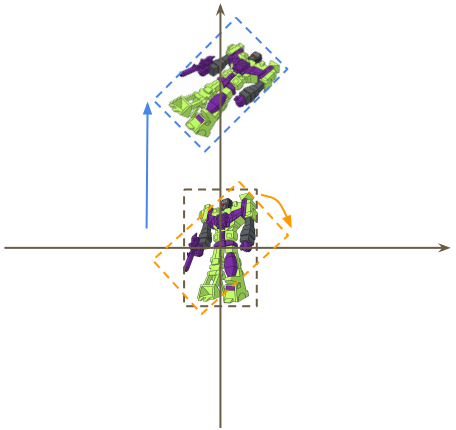
Image credit: Hasbro

Two different orders

Translate then **Rotate**



Rotate then **Translate**



Transforms in OpenGL

Now that we've reviewed all that maths

- We're now going to delegate the work to a library
- GLM applies the matrices for us
- We only rarely will have to manually enter values into a matrix or memorise Scale/Rotate/Translate

A small Case Study

With Textures and Transforms …

Is it possible to replicate Mario?

Image credit: Nintendo (but totally an artwork by Marc Chee)

Sprites (Textures)

We could use this image as a texture for Mario

- Change texture coordinates based on what actions we take
- Or what state Mario is in (mushroom or flower)
- This means Mario is just a rectangle with some code handling texture coords

Sprites of Mario
Image credit: Nintendo

Transforms for Mario

How do controls affect the character?

- Directional input:
  - Translate the character somewhere
  - Change the sprite to a running sequence?
  - Jumps might need special code
  - Wait, do we translate Mario or do we translate the whole world?
- Change of state:
  - Picking up a mushroom makes us scale Mario vertically to match the larger sprite
  - Do we scale the verts before or after we translate to the current position?

Environment

Sprites/Textures for the Environment

- Repeated textures for the ground
  - What kind of texture coordinates might we use for the ground under Mario?
  - What wrapping system would we use?
- How are we building the background?
  - Flat colour per level?
  - Individual objects with things like clouds or mountain textures on them?
  - Or one big sliding texture on a big rectangle?

What did we learn today?

Textures and Transforms

- Details on Texturing
- Using Linear Algebra to transform vertices
- A quick look at making 2D games