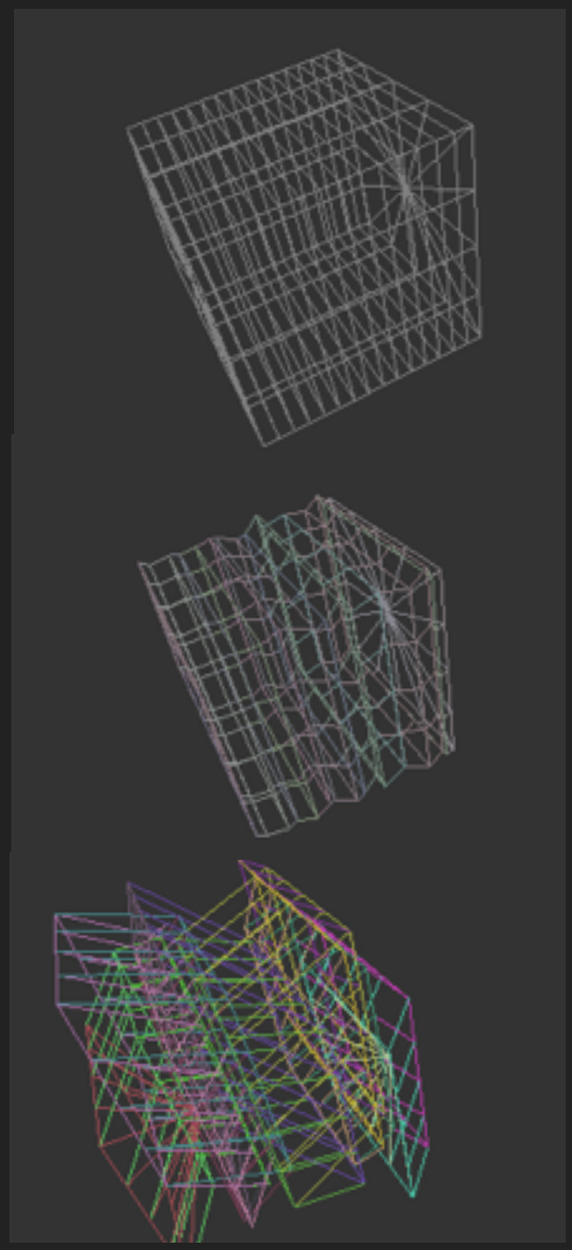
INT SYS 3

WEEK 5: INTRO TO 3D GRAPHICS AND OPENGL

INTRO: OPEN GRAPHICS LIBRARY (OPENGL)

- API for 3D and 2D vector graphics and shading/texturing operations
- It's a cross-platform API for a software interface for 3D graphics (and related graphics operations) hardware
- Put simply it is often used as an interface with Graphics Processing units
- Associated idea: bitmap graphics or video as textures on 3D objects; video handling can sometimes happen more efficiently on the GPU

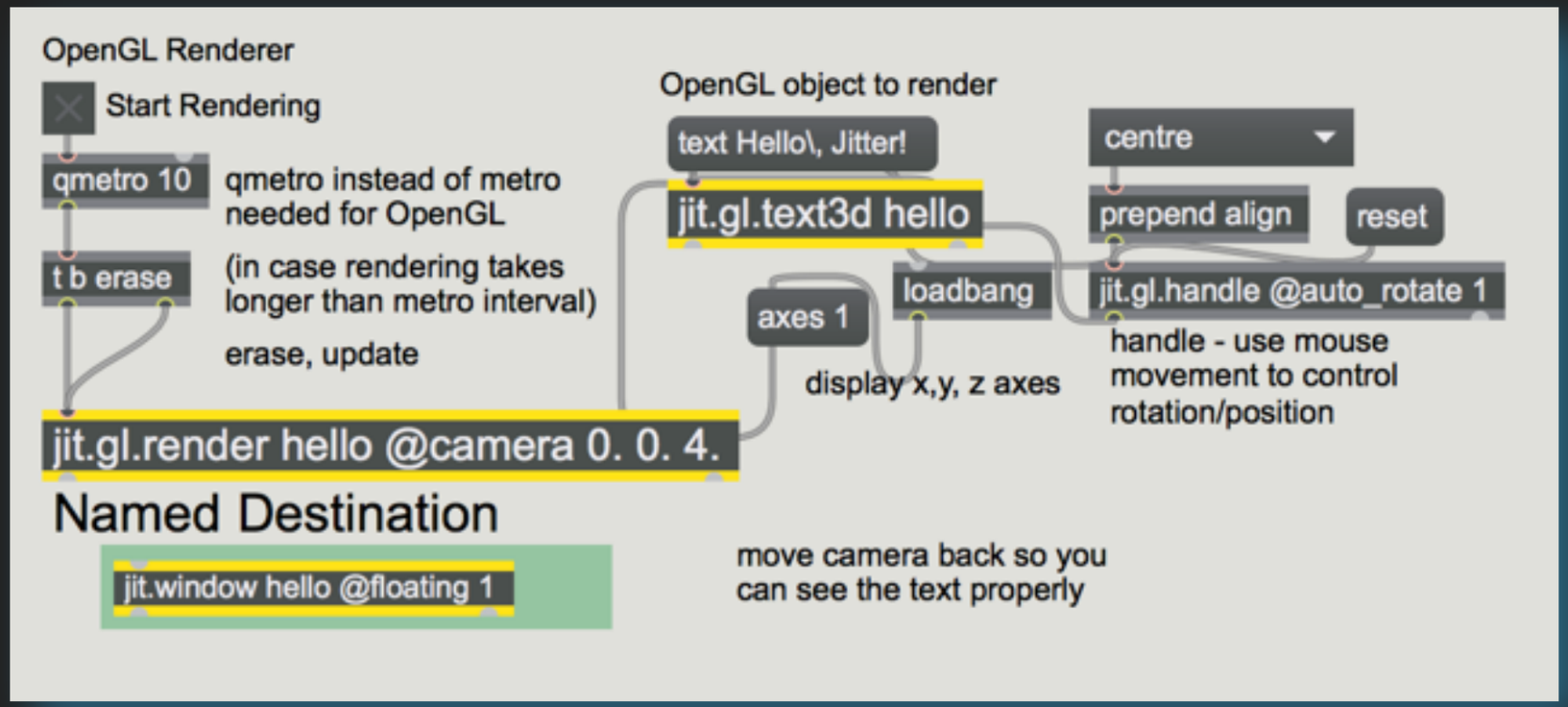


HISTORY, APPLICATIONS, VARIANTS

- OpenGL developed by Silicon Graphics in 1991, released 1992, but an open standard managed by a non-profit consortium
- Used in simulation, computer-aided design (CAD), games, etc.
- There are many variants, but one of the most prominent in recent years is WebGL (Open GL support within a browser via Javascript); WebGL consists of OpenGL commands in Javascript, plus some associated shader code directly addressed to the computer's GPU (controlling lighting, texture effects) in something called Open GL shading language (OGSL) which is C-like
- Shader: simplay a program that computes shading…texturing, lighting, effects; see more on shaders here (we won't go there just yet; maybe later!) https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313
- To see examples of WebGL-based experiments, look here: https://www.chromeexperiments.com/
- Another example is the 3D view available in some locations (and on powerful enough hardware) in Google Maps

OPENGL BASICS IN MAX

- From jit.(qt) to jit.gl!
- We need an OpenGL model/graphics object (we have families of OpenGL objects from basic geometric shapes to meshes to 2D and 3D text to ones which import 3D models)
- We need an OpenGL render object
- Remember this and you can't go too far wrong!



Q: what happens if we just send a bang message to jit.gl.render? (i.e. without 'erase' message)

We'll see in a moment! Let's play with the first 6 patches

JIT.GL.HANDLE

- Allows ‘click and drag’ control of OpenGL object positions
- Connect to object you want to control
- Also connect to named rendering destination (‘outout’ in this example) shared with jit.gl.render
- @auto_rotate 1 or 0
- Click and drag to spin object around various axes whilst its overall place is fixed
- To move around the 3D world, hold command for x/y movements, alt/option for z movements

CTRL is (presumably)
the substitute for CMD on Win

LIGHTING IN OPENGL

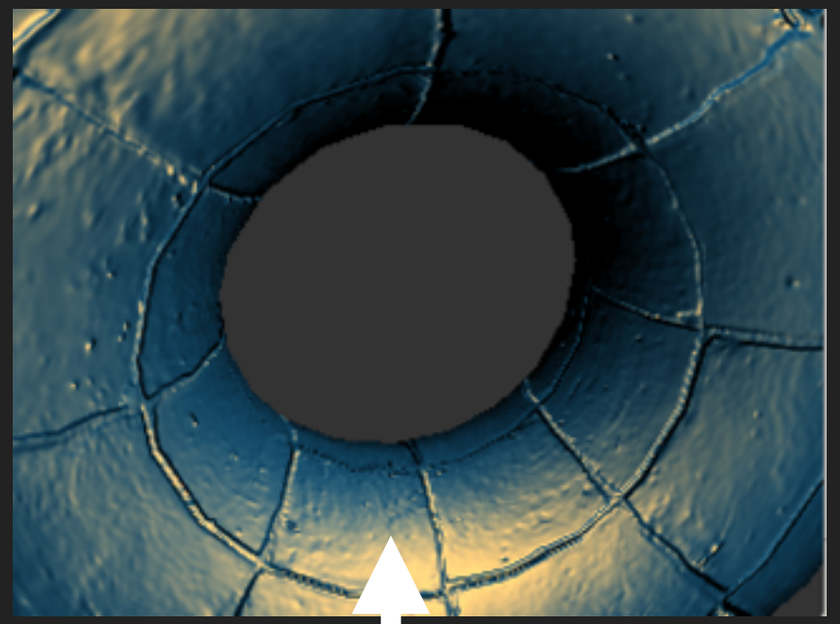
- OpenGL has a simple (but effective) model of real-world lighting
- It treats lighting in terms of three main components
- specular: particularly 'directional' light/'shininess'
- diffuse: directional influence at 'input', but scatters
- ambient light: directionless (light which bounces around in environment so that originating direction is unclear)
- PLUS emissive: for light sources
- Can specify colour values (RGB) for each component
- Not only this, we can specify how a material responds to incoming light by setting colour values for RGB for specular, diffuse, ambient

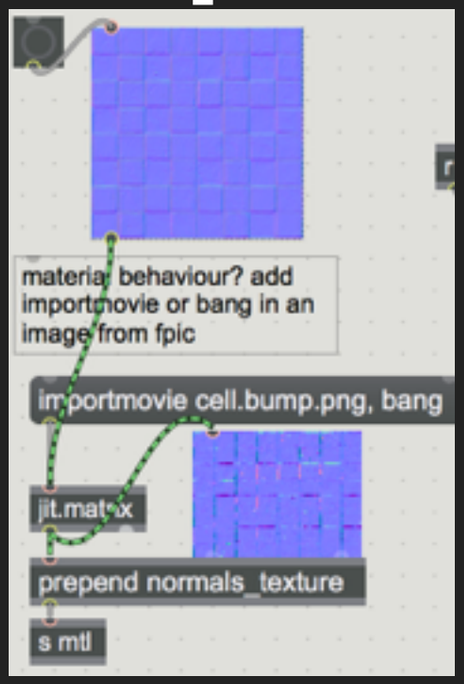
MORE ADVANCED OPENGL TEXTURING: MATERIALS

- Apart from simply applying a bitmap texture (as in patch 4), we can apply textures that effect how lighting behaves
- Jit.gl.material helps us do this; it can allow us to set material response to light (ambient, specular, diffuse)
- It can also set models for light behaviour, which alter how specular, diffuse behave (‘how shiny is shiny?)
- Bur, even more exciting, we can use bump maps

BUMP MAPS AND NORMAL MAPS

- Bump maps and normal maps change how light behaves when lighting is calculated, but don't otherwise alter the geometry of the 3D model. Repeat, normal geometry is unaffected!
- They're a good way of making a simple 3D model seem more detailed
- From Blender documentation:
- In CGI, bump maps can be applied to 3D models and thus serve to alter surface appearances. This can alter light interactions with scene objects. While bump maps can simulate topology such as surface bumps and ripples, it does not change the 3D geometry.
- Normal Maps and Bump Maps both serve the same purpose: they simulate the impression of a detailed 3D surface, by modifying the shading as if the surface had lots of small angles, rather than being completely flat. Because it's just modifying the shading of each pixel, this will not cast any shadows and will not obstruct other objects. If the camera angle is too flat to the surface, you will notice that the surface is not really shaped.
- Both bump maps and normal maps work by modifying the normal angle (the direction pointing perpendicular from a face), which influences how a pixel is shaded. (Normal maps variation of bump maps and term 'bump map' is sometimes interchangeably)
- Bump = displacement based on (simple) luminance, Normal = vector displacement based on RGB=xyz





Our example in jitter uses a normal map

HACKING OPENGL GEOMETRY

- We can output OpenGL object geometry data to a matrix; the matrix stores vertex data: the geometry of point locations which define the object
- This enables us to shape OpenGL geometry in powerful ways; e.g. we turn matrixoutput to 1 in a jit.gl.gridshape
- This matrix can now be changed using Jitter matrix processes before it goes to the jit.gl.render
- In our example, we can cross-fade it with jit.noise
- Jit.noise is set to output floating point values in a matrix which has 12 planes and 20 x 20 dimensions for each plane
- In our geometry matrix, we can have up to 13 planes. Planes 0-2 specify the x, y and z position of the vertex. Planes 3 and 4 specify the texture co-ordinates s and t. Planes 5-7 specify the normal vector nx, ny and nz used to calculate the effects of lighting on the geometry (normal mapping). Planes 8-11 specify the red, green, blue, and alpha vertex colour. Plane 12 specifies the edge flag e.
- (The output matrix of the jit.gl.gridshape object has 13 planes, but since we are not applying a texture to the geometry, and lighting is not enabled, the texture coordinates and normal vectors are ignored (we're also not using the edge flag, which is about connecting to another vertex)
- See tutorial 37 (Geometry under the hood) and OpenGL matrix format doc https://docs.cycling74.com/max5/tutorials/jit-tut/jitterappendixb.html for further details
- We'll continue with this idea in the next class

HACKING OPENGL GEOMETRY: SUMMARY

- Need to outputmatrix 1
- Can use jit.xfade to set amount matrix is affected by jitter source
- 13 planes, but if lighting, textures, and certain point connectsion aern't used, we have 7 effective dimensions (others just ignored)
- Planes 0-2 specify the x, y and z position of the vertex. Planes 8-11 specify the red, green, blue, and alpha vertex colour behaviours.
- => we can also truncate at 12 planes when creating matrix data to affect the geometry, hence the 12 planes for the jit.noise object