Abertay University

Vertex manipulation

CMP301 Graphics Programming with Shaders

This week

- The power of the vertex shader
  - Vertex manipulation
    - Displacement maps
    - Deforming surfaces
    - Morphing
- A simple example

Back to the vertex shader

- Been focusing of the fragment shader
  - There is more to come, but for now back to vertex shader
- Currently we do little with the vertex shader
- Vertex shader can be very powerful
  - More than just positioning geometry
- Why not do this first? It's the first stage in the pipeline!
  - Mainly because without lighting we won't see some of the stuff

Recap

- Vertex shader can manipulate properties such as
  - Position
  - Colour
  - Texture coordinates
  - Normals
- This must be done before rasterisation
- It can’t create new vertices

Vertex manipulation

- Most common manipulation we have already been doing
  - Applying transform matrices to vertex positions
  - Modifying the input position to reflect the placement of geometry within the scene
    - Translate, rotate, scale
  - Both position and normal vector define the “physical” representation of the model
    - Both are modified by the transformation matrices

Vertex manipulation

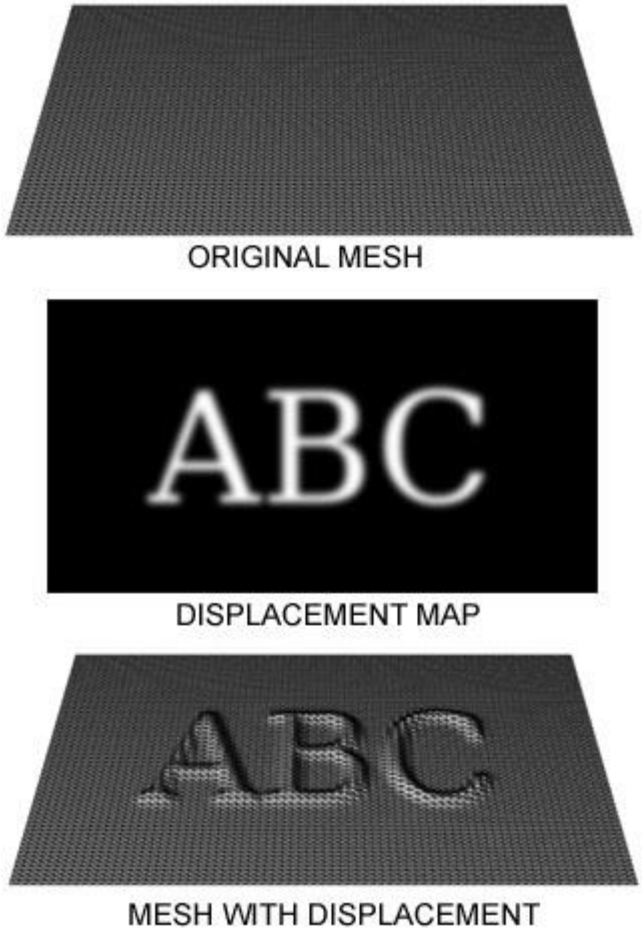
- We do this in the vertex shader to be certain the geometry has been converted into the desired coordinate space
- This has to be done before further processing later in the pipeline
  - Rasterisation

What can it do?

- Vertex manipulation
  - Including displacement maps
- Vertex morphing
  - Morph triangle meshes from one shape to another
- Mesh skinning
  - Animating meshes
- Deformation of surfaces
  - Create dynamic surfaces

Displacement maps

- Either a procedural or texture based approach
- Displacing the geometry based on data
- Usually along the vertex normal
- Adding depth and detail



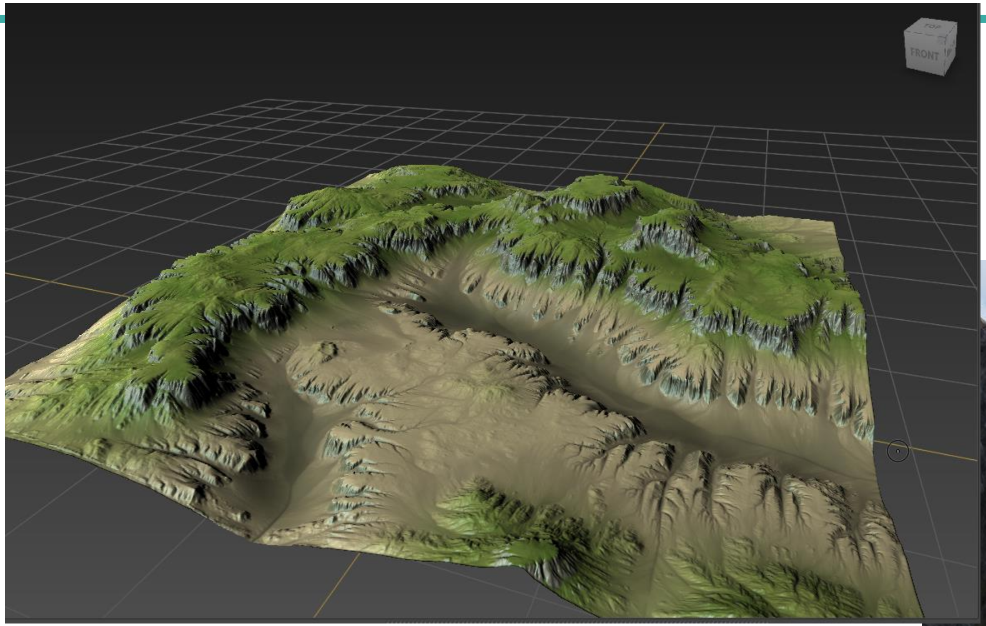
Displacement maps

- Widely used in terrain generation
  - Along with procedural approaches
  - You can use it to displace vertices on any shape
    - Creating seriously detailed and unique objects

Displacement maps

- There are also software packages designed to generate terrains
  - Using height maps (a form of displacement mapping)
- Most of these are packaged within Game Engines
- For example UE and CryEngine have some seriously powerful terrain generators
- Instead of having a massive sculpted 3D model using a high resolution mesh and a greyscale image you create it in real time
  - Usually combined with tessellation to improve performance

Some examples



Deformation of surfaces

- Very similar to displacement maps
- Key differences
  - No map
  - Usually dynamic
  - Can be controllable
  - Driven by interaction
- Largely controlled algorithmically

Deformation of surfaces

- For example
  - Pass a sine wave through a plane
  - Reposition the vertices (y coord) based on sin(x pos)
  - Flat plane becomes a lovely sine wave plane
- Same logic can be applied to other shapes
  - Cube
  - Sphere
  - Models
  - Etc
  - Doesn’t matter
- Add time to make the manipulation dynamic

A diagram

- Insert diagram here

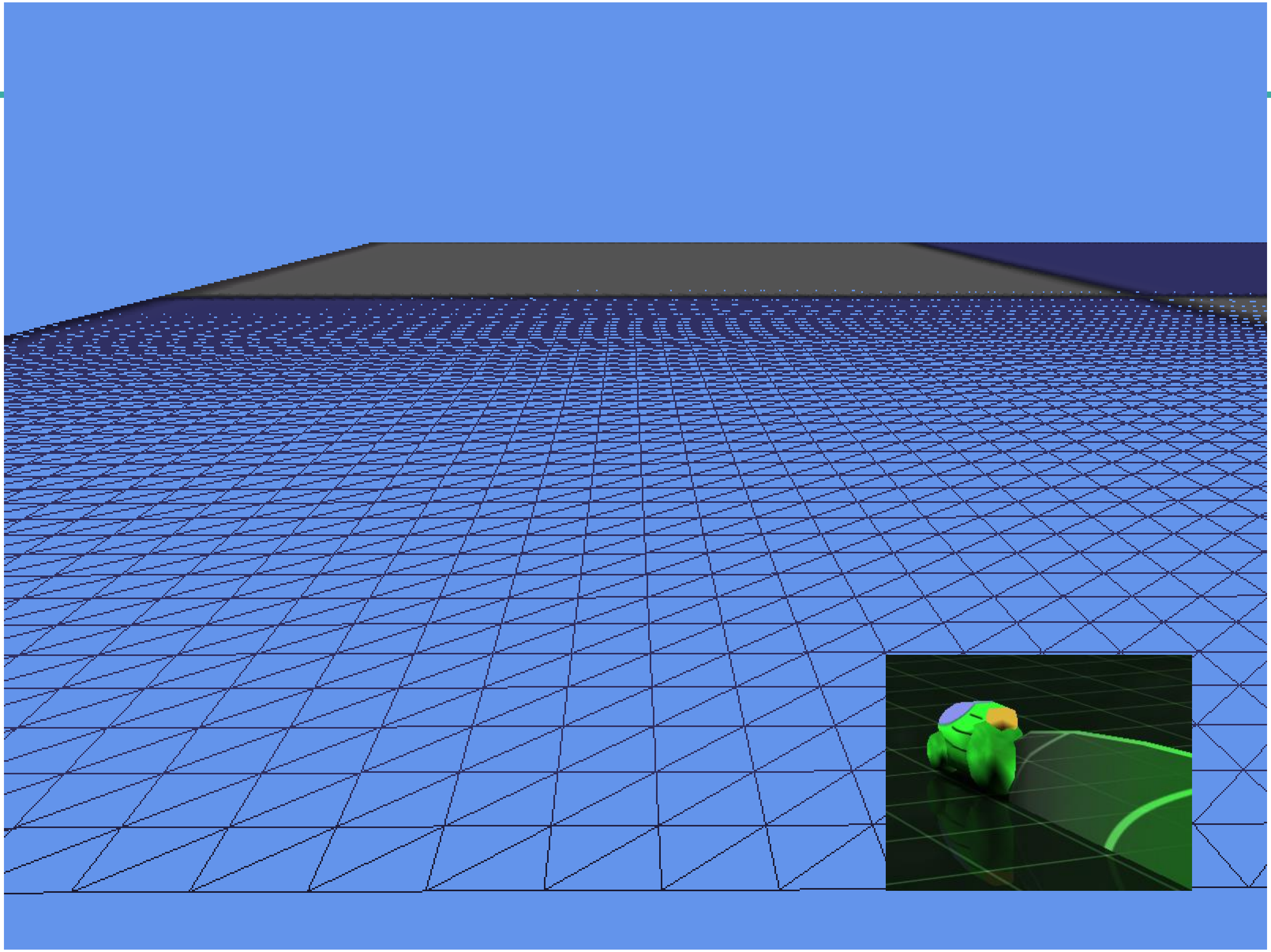
The rough idea

Mesh morphing

- Concept
  - Transform one mesh to another
  - Cube into sphere
  - Person into wolf
- Requires knowing position data for both shapes
  - Interpolate over time
- Requires the EXACT same number of vertices

A simple example

- Pass a sine wave through a plane mesh
- Requires a custom vertex shader to manipulate the vertices
- Plane mesh of “high” resolution
  - 100 by 100 quads
- Based on vertex position and time
  - Positional Y component will be modified
  - Via sine wave
- Moving the vertex is fairly straight forward
  - Calculating the normals ... not so much



Manipulation shader class

- Base functionality of a simple lighting shader
  - We will need some lighting to see what is happening
  - Some ambient and diffuse directional light will be fine
- Additional constant buffer to pass a time variable to the shaders

```
struct TimeBufferType
{
   float time;
   XMFLOAT3 padding;
};
```

  - The time variable SHOUD BE based on delta time from the timer class
  - Alternatively you can increment a counter every frame
    - If vsync is enabled the frame rate is clamped

Manipulation shader class

- You can pass additional data including
  - Height
  - Frequency
  - Etc
- Time controls the movement of the transform
- Height controls how much to offset each vertex

- This requires some modification of the vertex shader
  - Very little change to the pixel shader

Timer class

- The timer class is a simple class that calculates delta time
  - Already part of your project
  - Already used by the base application
  - Only a couple functions
    - Initialise()
    - Frame()
    - GetTime()
- Application has access and during each frame has the timer update it’s internal counter
- We will need this for calculating vertex movement

Vertex shader

```
OutputType main(InputType input)
{
    OutputType output;
    float height = 1.0f;

    // Change the position vector to be 4 units for proper matrix calculations.
    input.position.w = 1.0f;

    //offset position based on sine wave
    input.position.y = height *sin(input.position.x + time);

    //modify normals
    input.normal.x = 1 - cos(input.position.x + time);
    input.normal.y = abs(cos(input.position.x + time));

    // Calculate the position of the vertex against the world, view, and projection matrices.
    output.position = mul(input.position, worldMatrix);
    output.position = mul(output.position, viewMatrix);
    output.position = mul(output.position, projectionMatrix);


    . . .
```

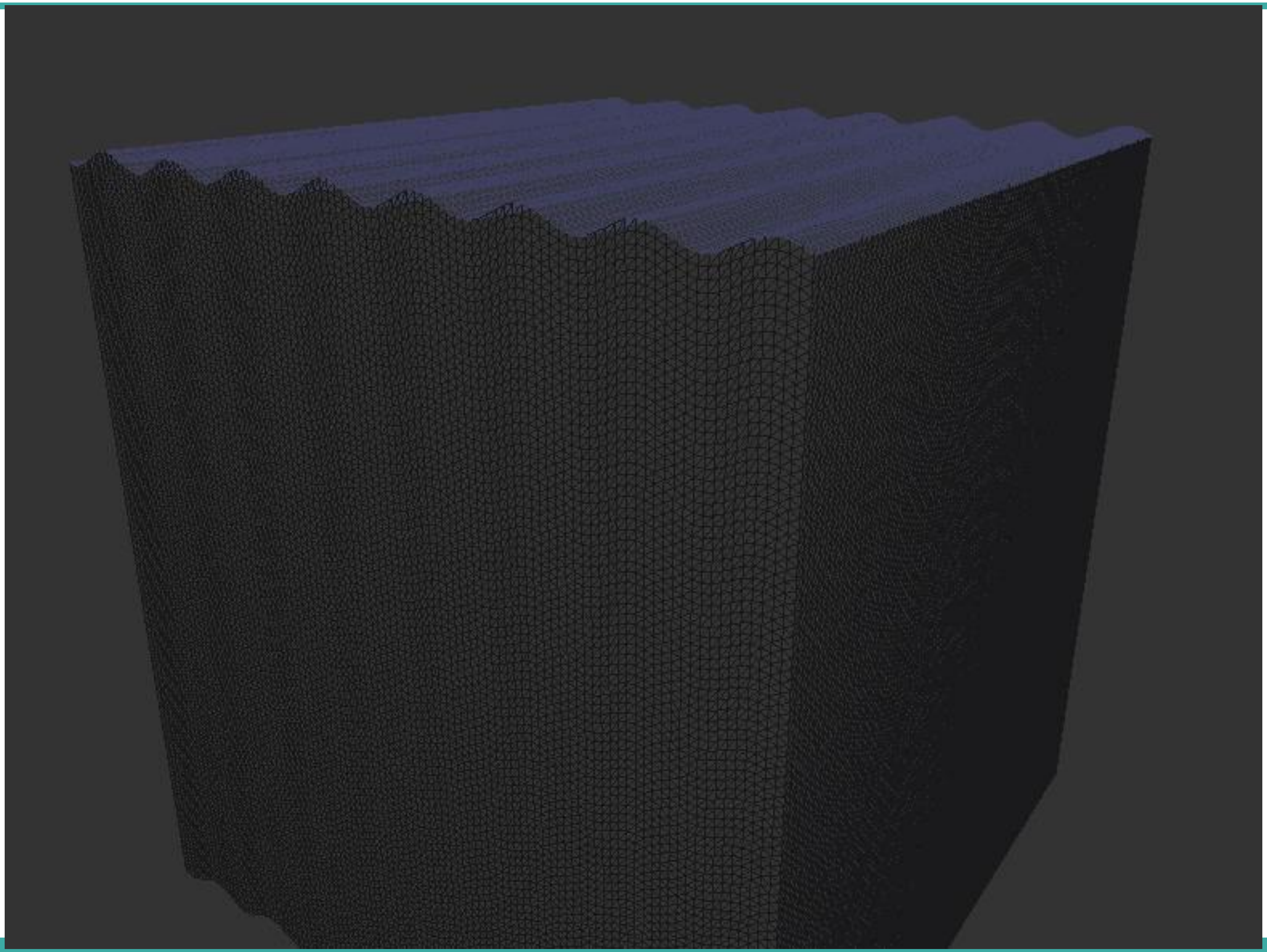
Example

In wireframe

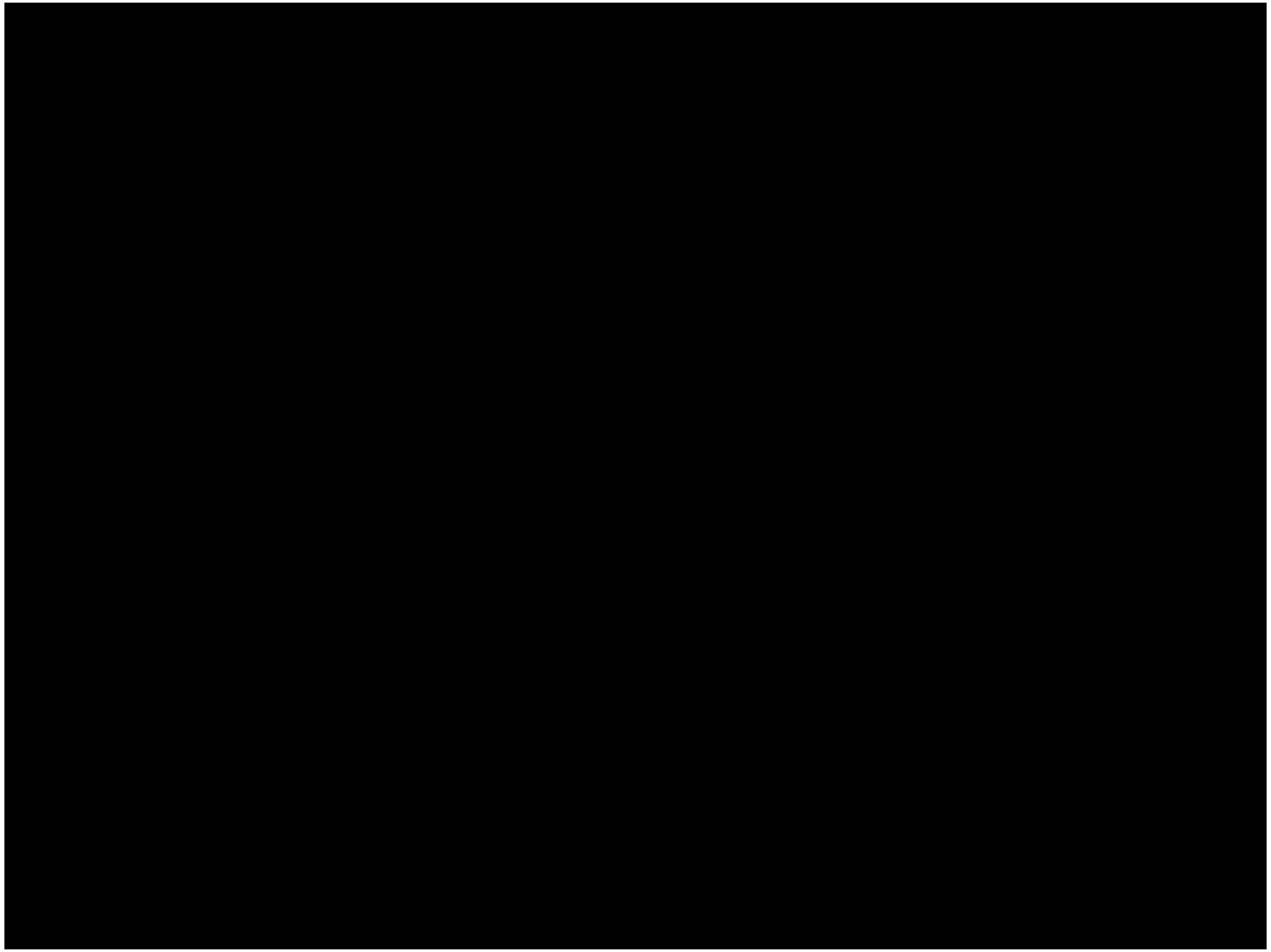
More examples

- The classic example is the Jelly cube
  - Using the same sine wave offset
- And something else I made

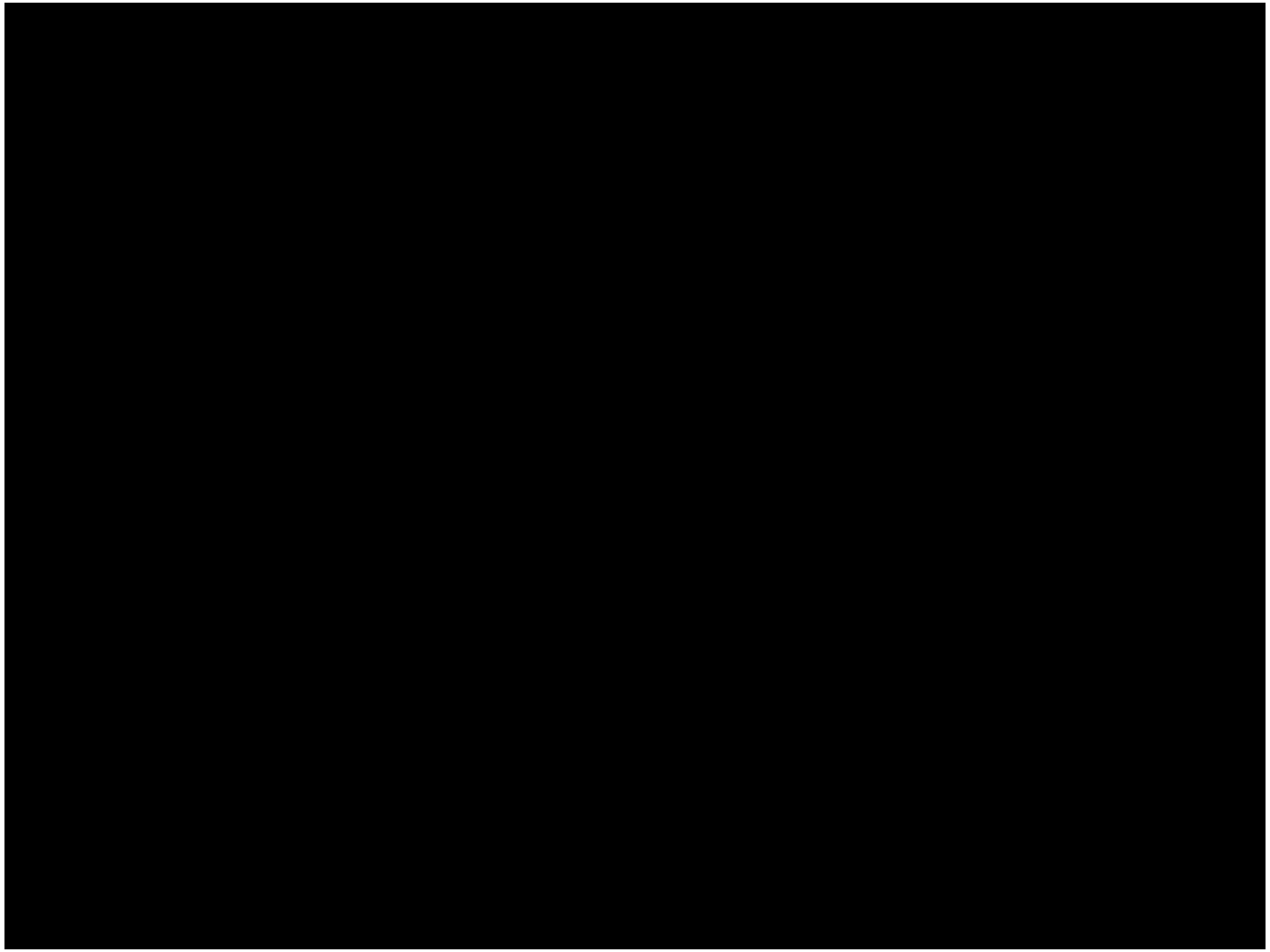
Jelly cube



Alternative equation



Mesh morphing



In the labs

- Sine wave through a plane
- Other vertex manipulation