

Week 11

Lighting, Physics and Optimisation

Vector Variable in C++

Vectors in C++

```
std::vector<type> vec;  
vec.push_back(1);  
vec.push_back(2);  
vec.resize(4);  
vec.pop_back();
```

The diagram illustrates the state of a C++ vector after the provided code. It shows three stages of the vector's state:

- Initial state: A vector containing elements 1 and 2.
- After `vec.push_back(1);` and `vec.push_back(2);`: The vector contains elements 1, 2, and two empty slots (Empty Empty).
- After `vec.resize(4);` and `vec.pop_back();`: The vector contains element 1, and three empty slots (Empty Empty Empty).

https://www.youtube.com/watch?v=H1cK1QXw6Zs&ab_channel=NesoAcademy

Lighting in OpenGL

Reading Data

- *PLY Face Elements:*
 - *Supports free form polygons*
 - *(common are triangles, quads)*
 - *Vertex textures can be empty*
 - *To use in OpenGL convert everything to Triangless*

element vertex 386

property float x

property float y

property float z

property float nx

property float ny

property float nz

property uchar red

property uchar green

property uchar blue

property uchar alpha

element face 768

property list uchar int vertex_indices

end_header

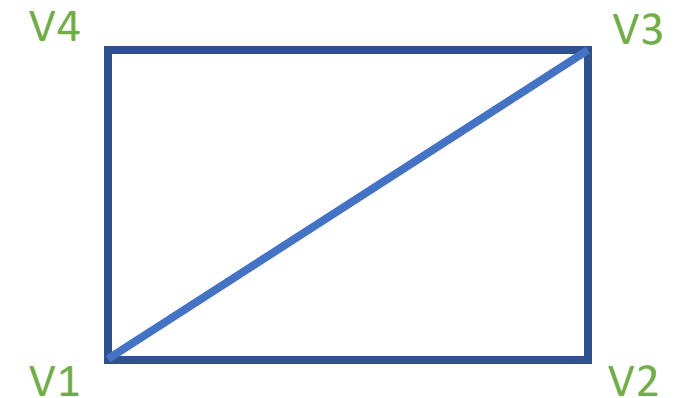
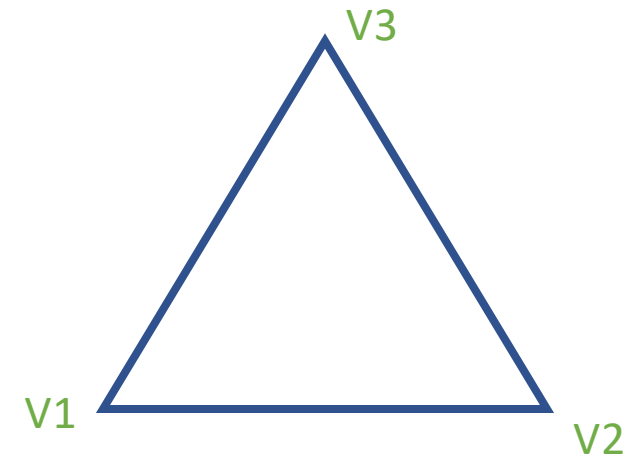
0.5 0.5000001 -0.4999999 0.5773503 0.5773504 -0.5773501 97 138 70 255

0.5 -0.4999999 -0.5000001 0.5773503 -0.5773501 -0.5773505 97 138 70 255

3 98 99 100

3 101 102 103

3 104 105 106



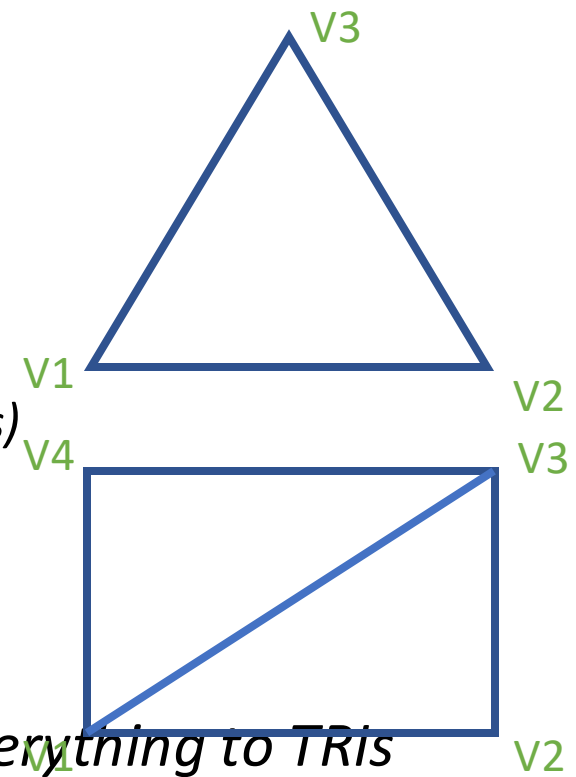
Reading Data

```
element vertex 386
property float x
property float y
property float z
property float nx
property float ny
property float nz
property uchar red
property uchar green
property uchar blue
property uchar alpha
```

element face 768

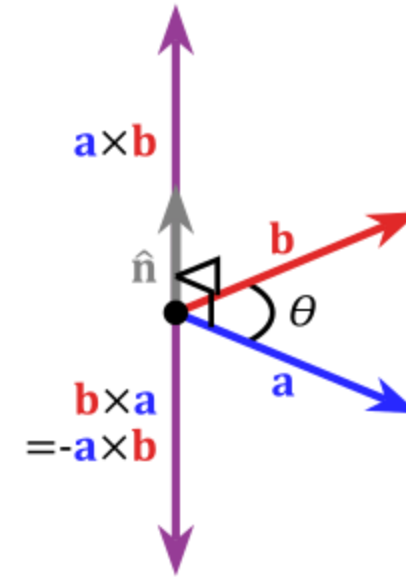
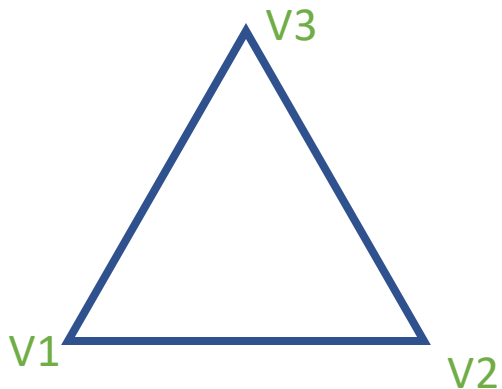
```
property list uchar int vertex_indices
end_header
0.5 0.5000001 -0.4999999 0.5773503 0.5773504 -0.5773501 97 138 70 255
0.5 -0.4999999 -0.5000001 0.5773503 -0.5773501 -0.5773505 97 138 70 255
3 98 99 100
3 101 102 103
3 104 105 106
```

- *PLY Face Elements:*
 - *Supports free form polygons*
 - *(common are triangles, quads)*
 - *Vertex textures ,*
 - *Normals, ([link](#))*
 - *Colour*
 - *To use in OpenGL convert everything to TRIs*



Catch-UP

- *Normal vectors:*
 - *Perpendicular to a face*
 - *Identifies which side the polygon is facing*
 - *Should be unit vectors $||v|| = 1$*
 - *Can be calculated using cross product*



$$\begin{array}{l}
 +i u_2 v_3 \\
 +u_1 v_2 k \\
 +v_1 j u_3 \\
 -v_1 u_2 k \\
 -i v_2 u_3 \\
 -u_1 j v_3
 \end{array}
 \begin{vmatrix}
 \mathbf{i} & \mathbf{j} & \mathbf{k} \\
 u_1 & u_2 & u_3 \\
 v_1 & v_2 & v_3
 \end{vmatrix}$$

Reading Data

element vertex 30963

property float x

property float y

property float z

property float nx

property float ny

property float nz

property float **s**

property float **t**

element face 39536

property list uchar uint vertex_indices

end_header

9.224723 8.123055 22.765438 0.586779 -0.597766 0.546190 0.000000

0.000000

6.038172 8.422766 14.462516 -0.587848 -0.556169 -0.587420 1.000000 -

1.000000

- *DAE Face Elements:*

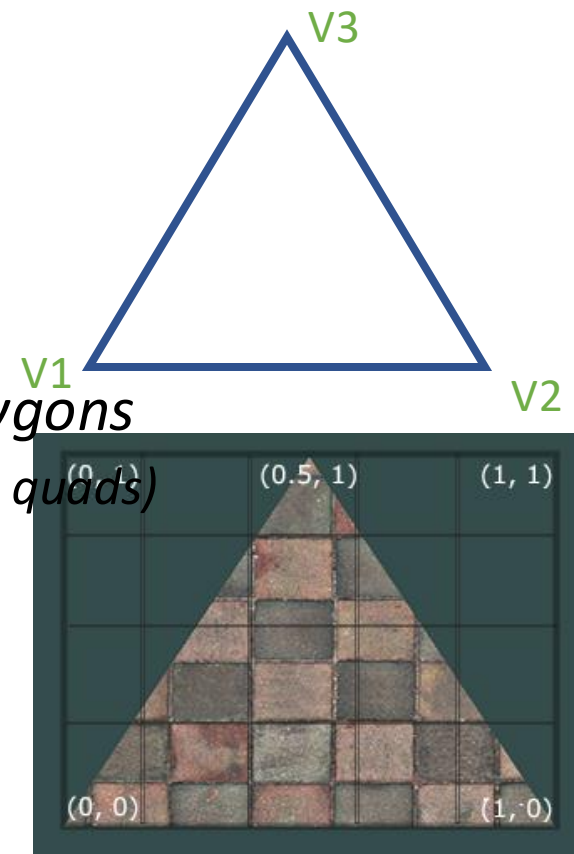
- *Supports free form polygons*

- *(common are triangles, quads)*

- *Vertex textures* , ([Link](#))

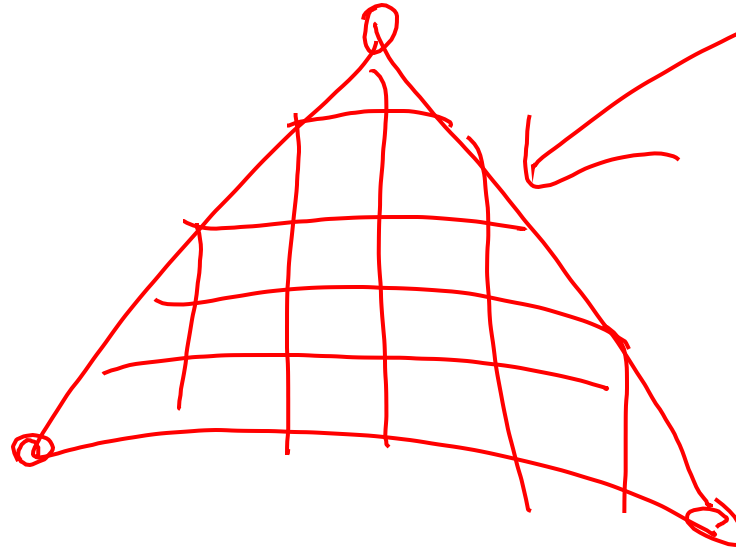
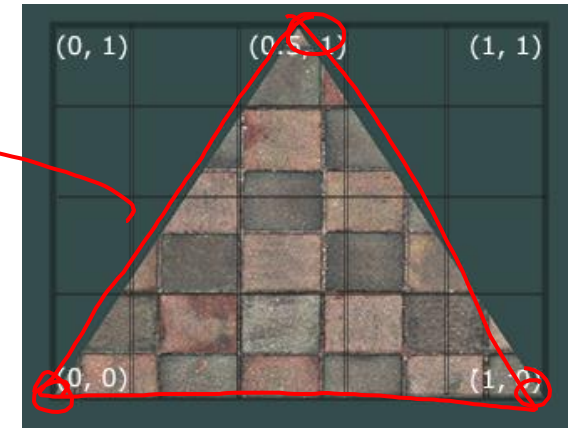
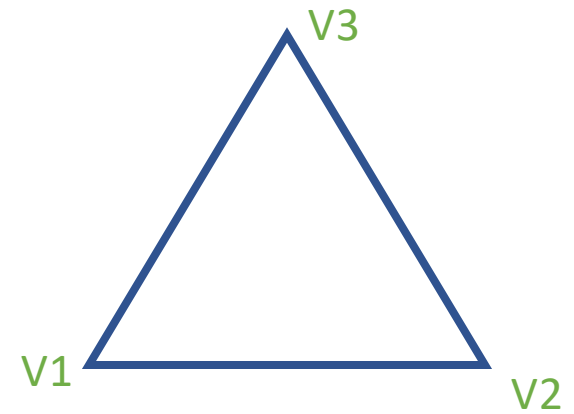
- *Normals*,

- *Colour*



Reading Data

- *UV Coordinate Face Elements:*
 - *Supports free form polygons*
 - *(common are triangles, quads)*
 - *Vertex Mapping*

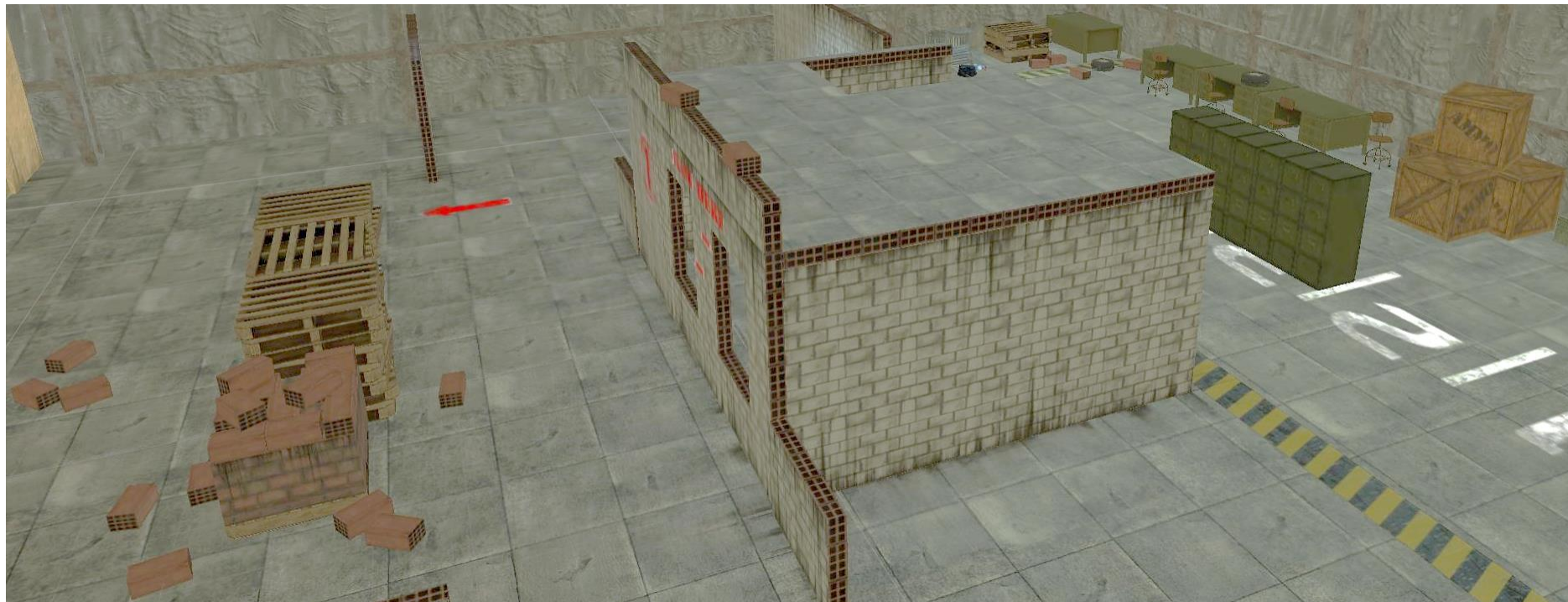


Light and Matter

- *We have:*
 - *Geometry/Movement*
 - *Perspective*
 - *Simple Colour/Texture*
- *We are missing:*
 - *Light (added depth perception/realism/immersion)*
 - *Sound (maybe covered later)*
 - *Control (basics are simple follow learnopengl and extension of Project2)*

Light and Matter

- *Is light really important?*



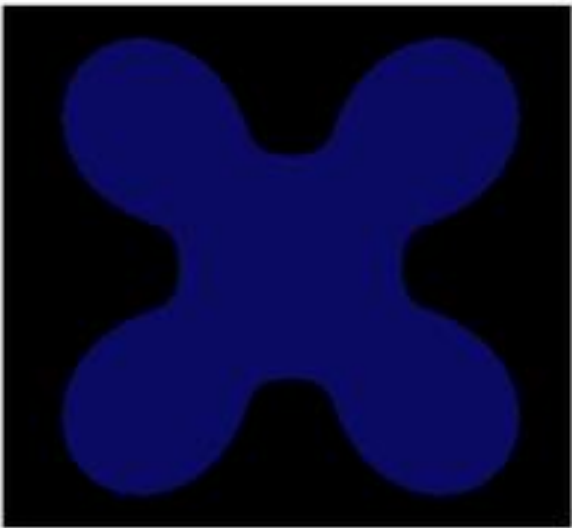
Light and Matter

- *Is light really important?*

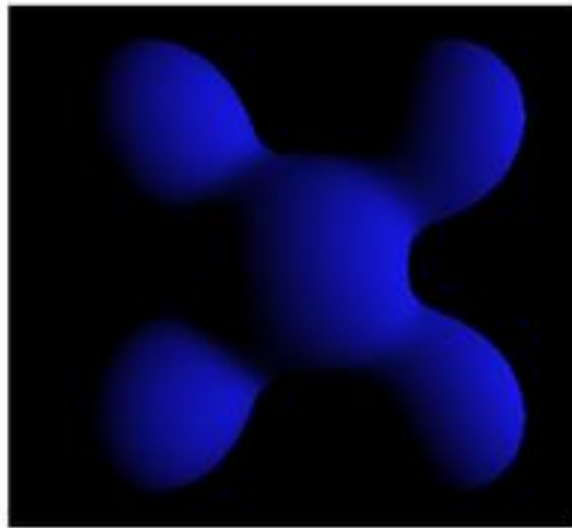


Light and Matter (Phong Lighting Model)

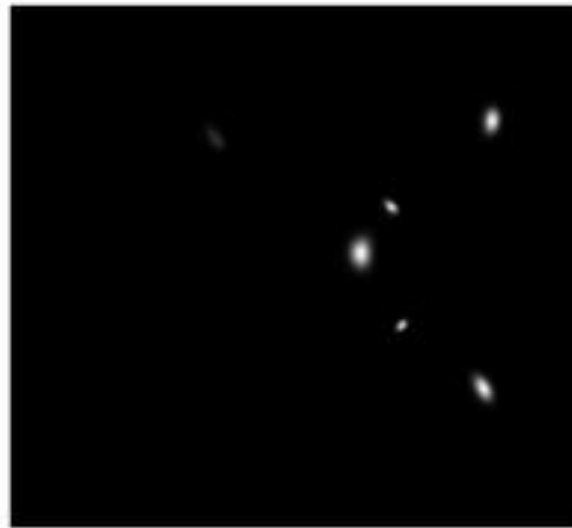
- Is light really important?*



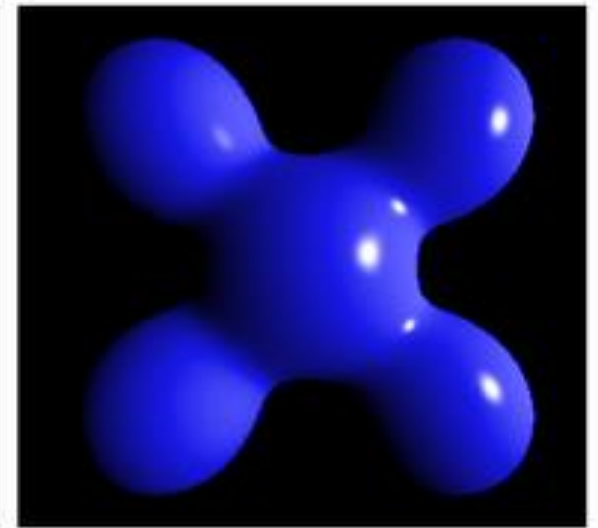
ambient



diffuse

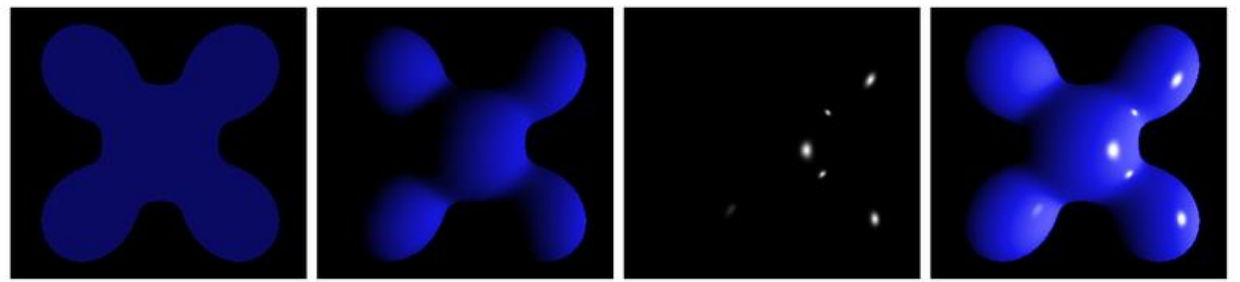


specular



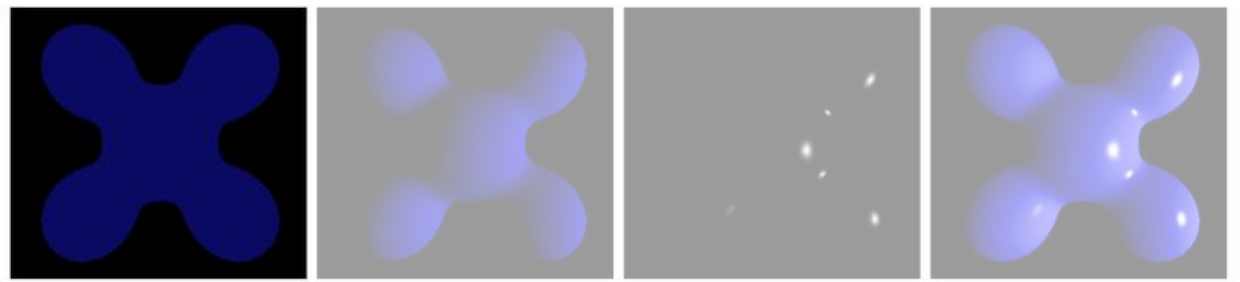
*ambient +
diffuse +
specular*

Light and Matter



- *Adding A Light Source(Ambient Lighting)*
- *Adding Diffuse Light (Basic Shading)*
- *Adding Specular Light (Materials)*
- *Combing Everything*
 - *When used in the vertex shader (Gouraud Shading)*
 - *Still exposed rough geometry, visible in specular aspect*
 - *When used in the fragment shader (Phong Shading)*
 - *Normals are interpolated/smoothened*

Ambient Light

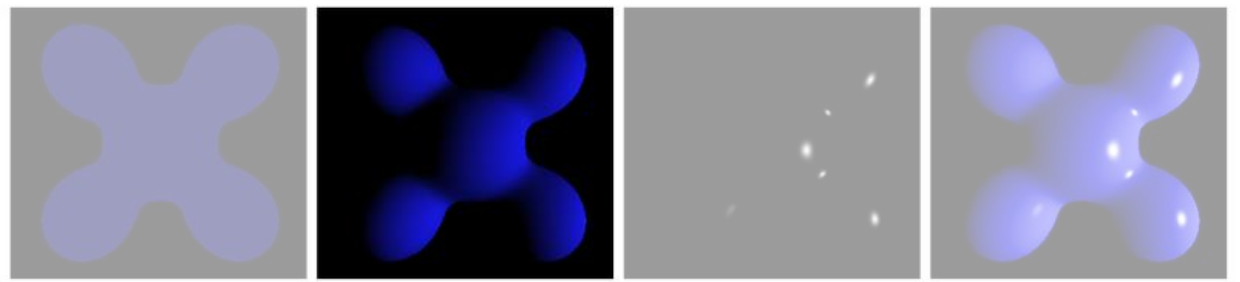


- *Adding A Light Source(Ambient Lighting)*

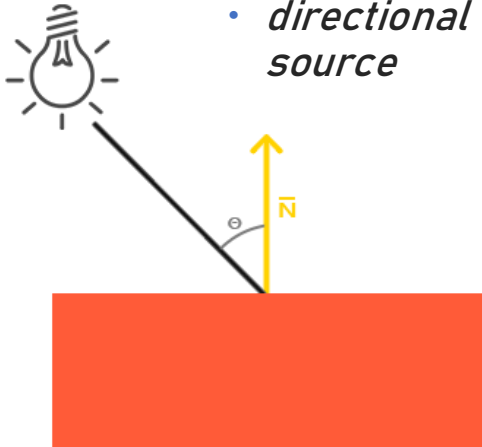
```
// ambient light
glm::vec4 ambient = glm::vec4(0.1f, 0.1f, 0.1f, 1.0f);
//adding the Uniform to the shader
int aLoc = glGetUniformLocation(program, "ambient");
if (aLoc)
glUniform4fv(aLoc, 1, glm::value_ptr(ambient));
```

```
//vert shader
fragColour = ambient * vColour;
```


Diffuse Light



- *Adding Diffuse Light (Basic Shading)*
 - *directional component of a specific light source*

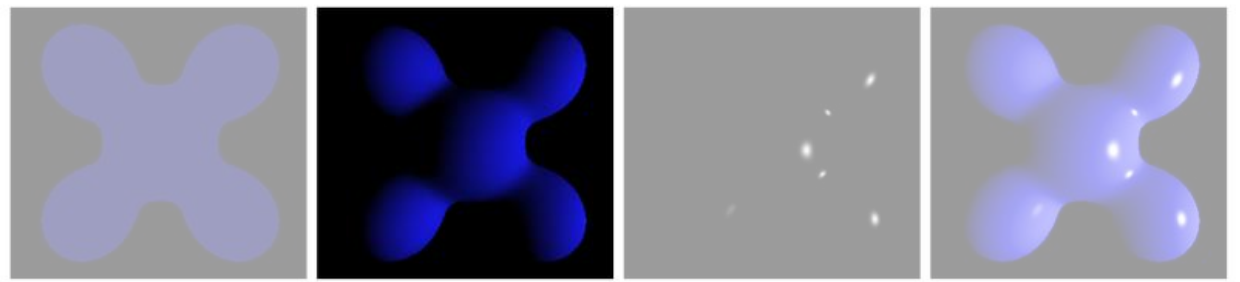


- *uses:*
 - *diffuse light component*
 - *normal vector of the face*
 - *Requires a light (location) in space*
- *creates a more natural representation*

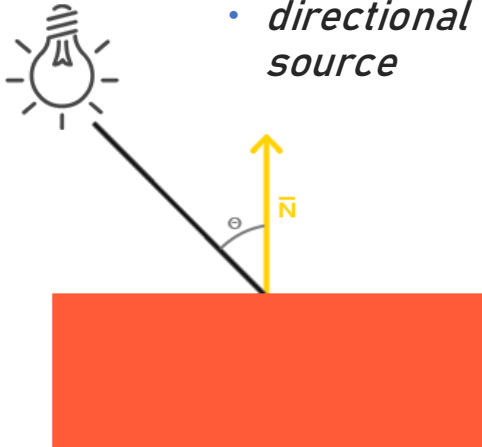
```
// light object
glm::vec3 lightPos = glm::vec3(10.0f, 25.0f, 20.0f);
GLuint dLightPosLoc = glGetUniformLocation(shader, "lightPos");
glUniform3fv(dLightPosLoc, 1, glm::value_ptr(lightPos));

// diffuse light
glm::vec3 diffuseLight = glm::vec3(0.5f, 0.5f, 0.5f);
GLuint dLightLoc = glGetUniformLocation(shader, "dLight");
glUniform3fv(dLightLoc, 1, glm::value_ptr(diffuseLight));
```

Diffuse Light



- *Adding Diffuse Light (Basic Shading)*
 - *directional component of a specific light source*



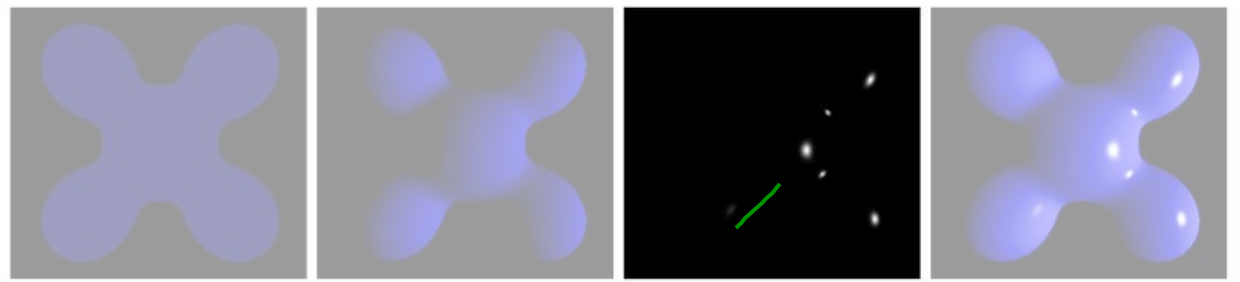
- *uses:*
 - *diffuse light component*
 - *normal vector of the face*
 - *Requires a light (location) in space*
- *creates a more natural representation*

```
// light object
glm::vec3 lightPos = glm::vec3(10.0f, 25.0f, 20.0f);

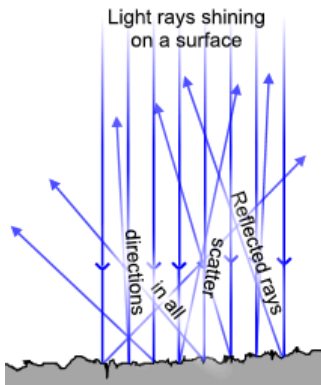
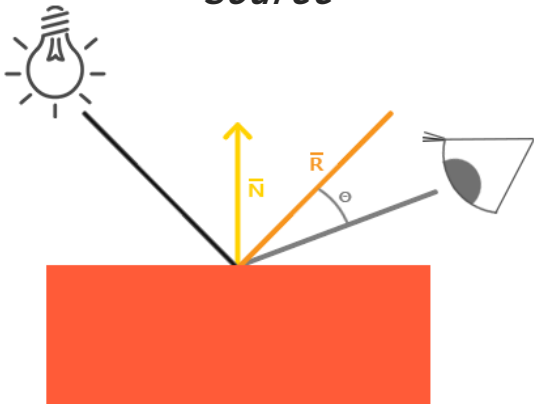
// diffuse light
glm::vec3 diffuseLight = glm::vec3(0.5f, 0.5f, 0.5f);
```

```
Void main() {
    gl_Position = mvp * vec4 (vPosition,1.0);
    float fLightScale = max(0.0, dot(vNormal,lightPos));
    vec3 vDiffuseColour = dLight * fLightScale;
    fragColour = (ambient + vec4(dLight,1.0) ) * vColour;
... }
```


Specular Light



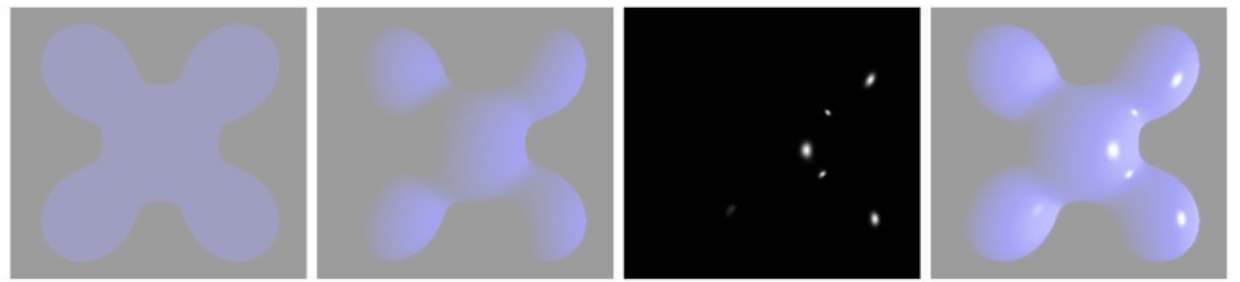
- *Adding Specular Light (Material)*
 - *Adding highlight and reflection from the light source*



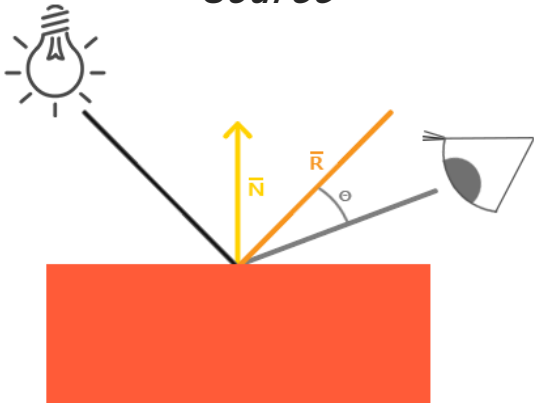
- *uses:*
 - *specular light component*
 - *normal vector of the face*
 - *Shininess of the material*
 - *Requires a light (location) in space*
- *creates a more natural representation*

```
// specular light
glm::vec3 specularLight = glm::vec3(0.2f);
GLfloat shininess = 128; //128 is max value
GLuint sLightLoc = glGetUniformLocation(shader, "sLight");
GLuint sShineLoc = glGetUniformLocation(shader, "sShine");
glUniform3fv(sLightLoc, 1, glm::value_ptr(specularLight));
glUniform1fv(sShineLoc, 1, &shininess);
```

Specular Light



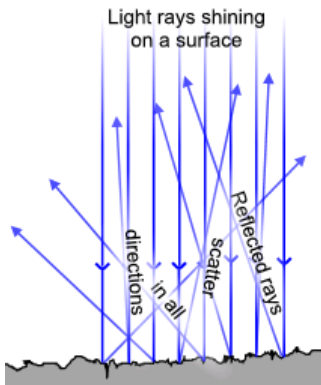
- *Adding Specular Light (Material)*
 - *Adding highlight and reflection from the light source*



- *uses:*
 - *specular light component*
 - *normal vector of the face*
 - *Shininess of the material*
 - *Requires a light (location) in space*
- *creates a more natural representation*

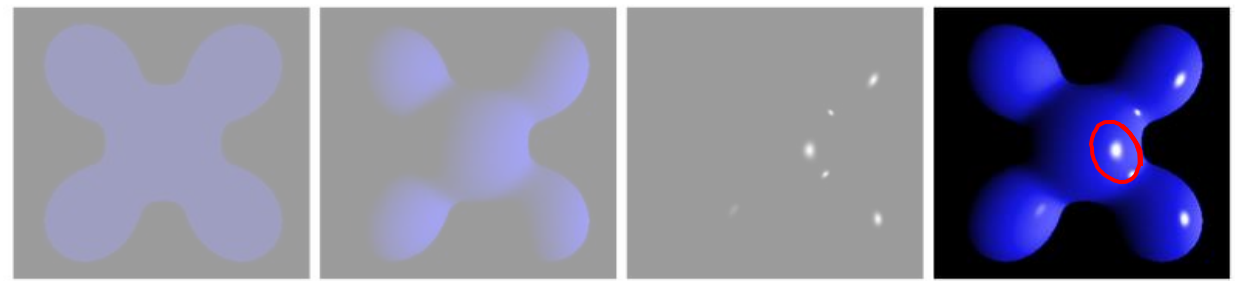
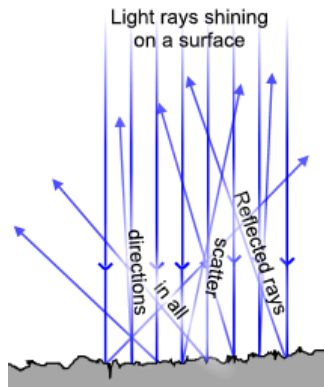
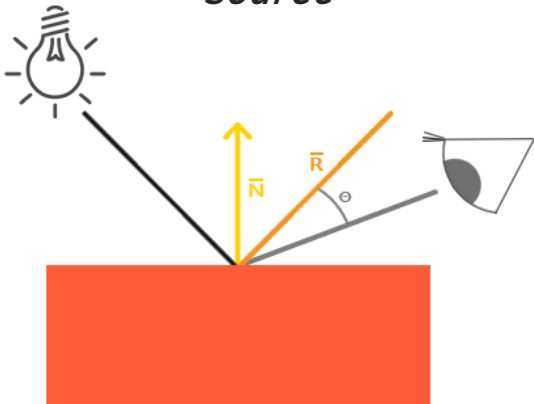
```
// specular light  
glm::vec3 specularLight = glm::vec3(0.2f);
```

```
GLfloat shininess = 128;
```



Specular Light

- *Adding Specular Light (Material)*
 - *Adding highlight and reflection from the light source*



```
Void main(){
    // view-space coordinate
    vec4 P = mv_matrix * vec4(vPosition,1.0);
    // calculate normal in view-space
    vec3 N = mat3(mv_matrix) * vNormal;
    // calculate light vector in view space
    vec3 L = lightPos - P.xyz;
    // calc the view vector
    vec3 V = -P.xyz;

    //Normalise
    N = normalize(N);
    L = normalize(L);
    V = normalize(V);

    // calc the reflection on the plane for the normal
    vec3 R = reflect(-L,N);

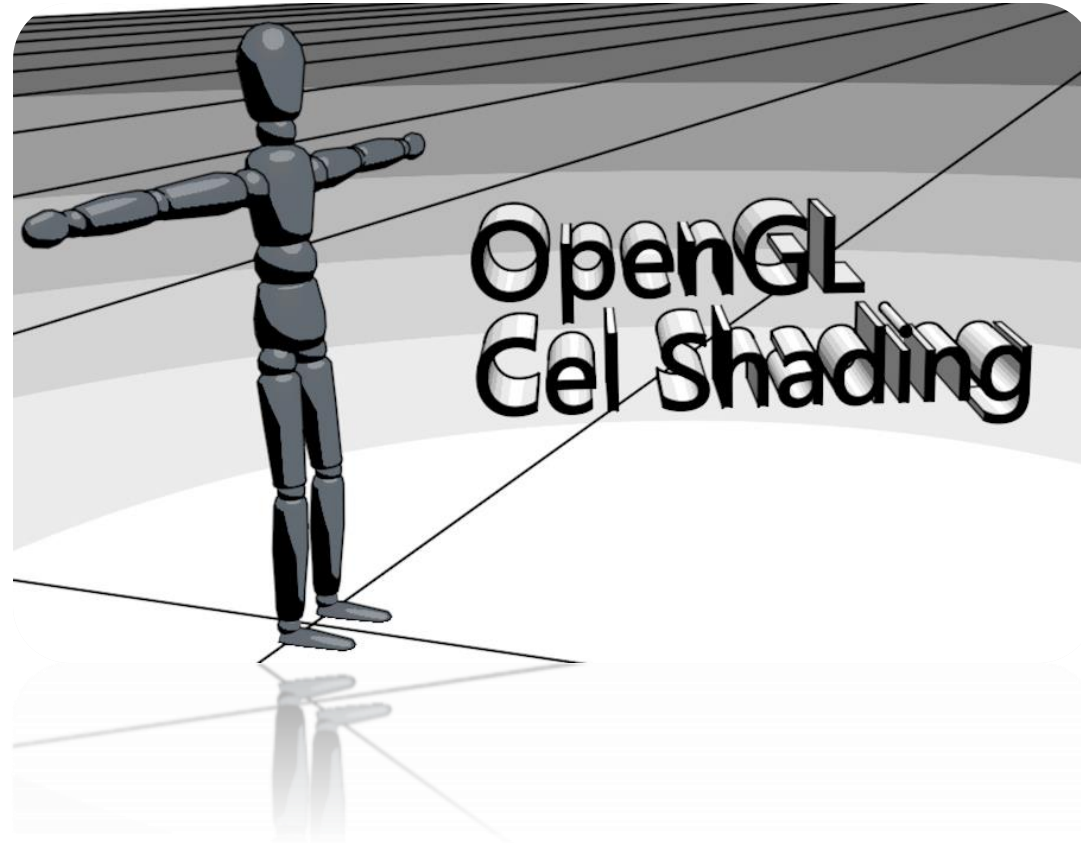
    // added specular and diffuse contributions to the
    colour

    vec3 diffuse = max(0.0, dot(N,L)) * dLight;
    vec3 specular = pow(max(dot(R,V), 0.0), sShine) *
    sLight;

    FragColour = (ambient + vec4(diffuse,1.0) +
    vec4(specular,1.0) ) * vColour; }
```

Advanced Effects

- [Cel Shading \[1\]](#)



Changing Outline & Gradient

- *Standard unlit image*



Changing Outline & Gradient

- *Strengthened outline & gradient texture*



Physics

--

Topics for this lecture

- *Introduction to Newtonian Physics*
- *Physics & PCG*

Physics in Games

- not all games need physics
- Games based on real world dynamics should look realistic (action <-> reaction)
- More complex games need more physics:
 - sliding through a turn in a race/car, sports games, flight simulation, etc.
 - Running and jumping off the edge of a cliff
- Two types of physics:
 - Elastic, rigid-body physics, $F = ma$, e.g., pong
 - Non-elastic, physics with deformation: clothes, chain, hair, volcanoes, liquid, boomerang
- Elastic physics is easier to get right

Physics in Games

- Modelling the movement of objects with velocity
- Where is an object at any time t ?
- Assume distance unit is in pixels
- Position at time t for an object moving at velocity v , from starting position x_0 :
 - $x(t) = x_0 + v_x t$
 - $y(t) = y_0 + v_y t$
- Incremental computation per frame, assuming constant time step and no acceleration:
 - v_x and v_y constants, pre-compute
 - $x += v_x, y += v_y$

Newtonian Physics (Three Laws of Motion)

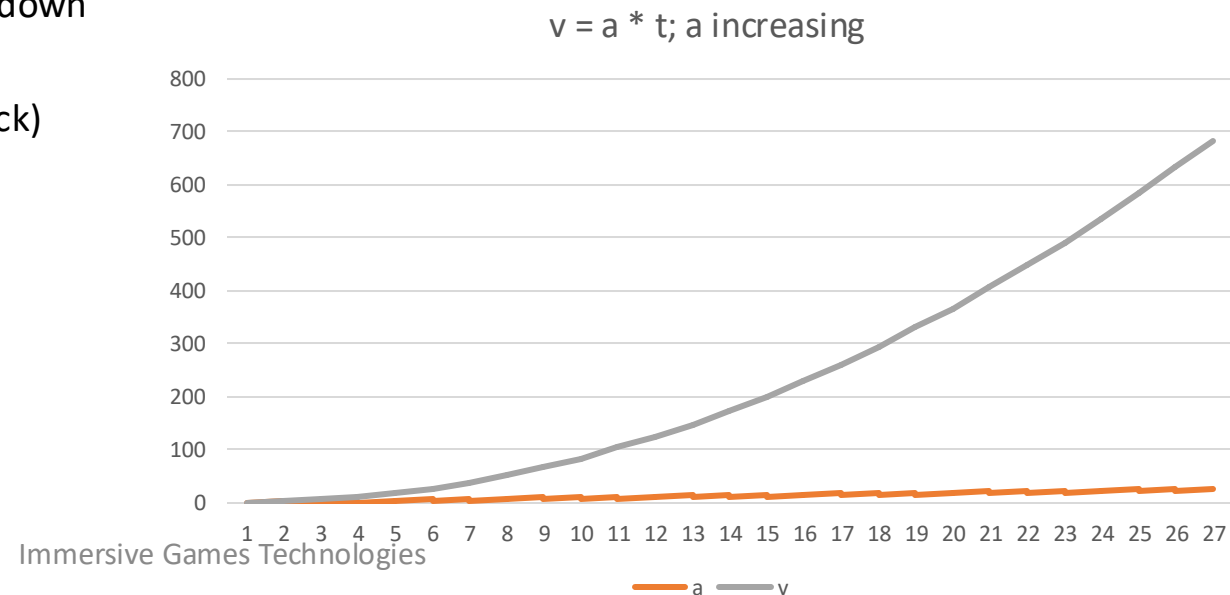
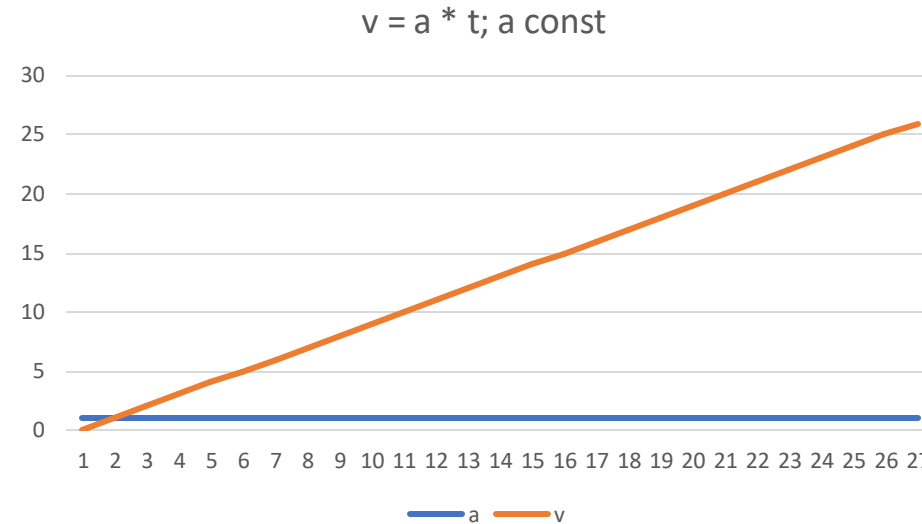


- **Inertia:** If no force is applied on an object, its velocity (speed and direction of motion) shall not change.
- **Force, Mass, and Acceleration:** The force acting on an object is equal to the mass of the object multiplied by its acceleration (rate of change of velocity). This is given by the formula $F = ma$.
- **Action and Reaction:** “For every action there is an equal and opposite reaction.” In other words, whenever one body exerts a force on another, the second body exerts a force of the same magnitude and opposite direction on the first.

Newtonian Physics (Acceleration)

velocity: $v = \frac{dx}{dt}$

- change of position over time
- acceleration applied over a given time
- $v = a * t$
- cars, people, rockets ...
 - changing acceleration: speed up, slow down
- bullets, ...
 - assume constant acceleration (dirty trick)



Newtonian Physics

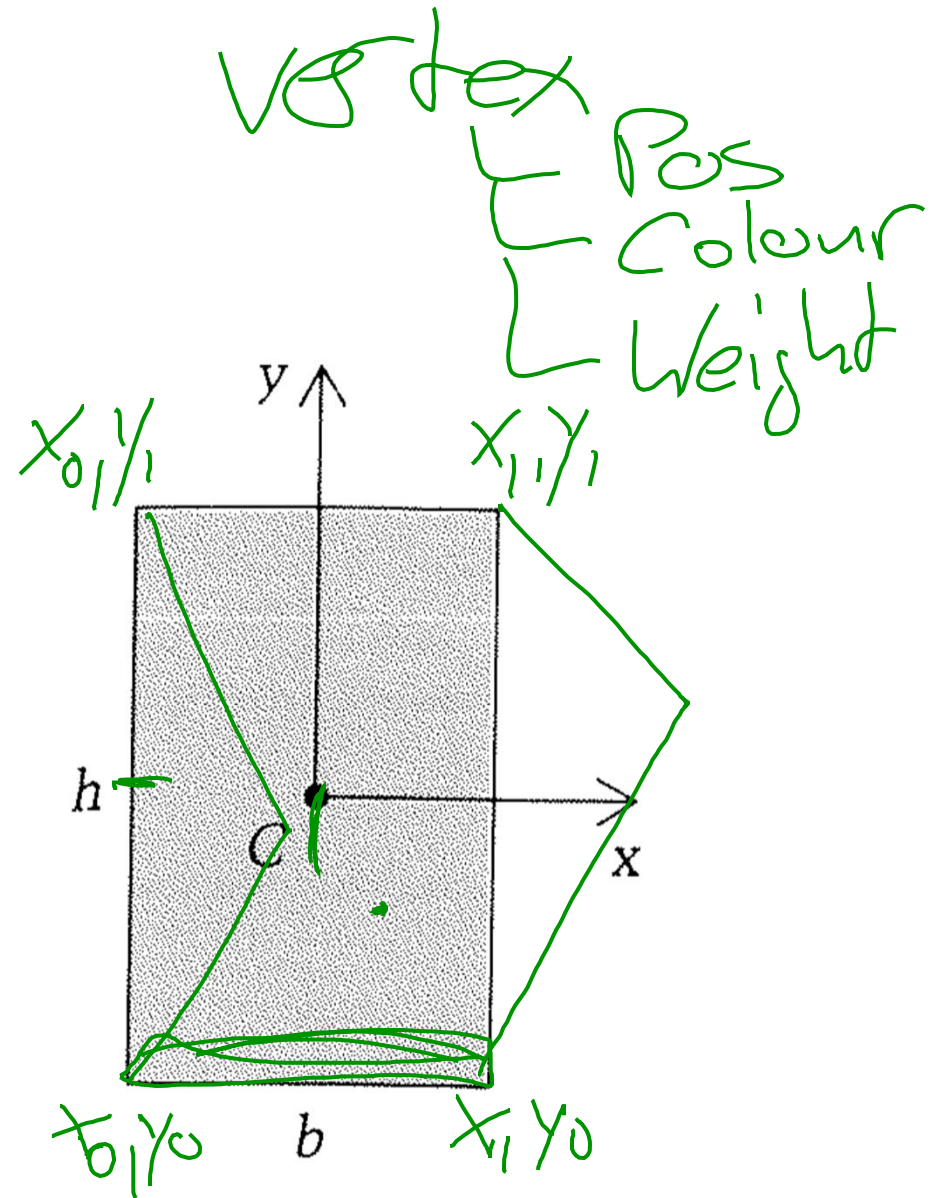
(Gravity)

- Gravity is a force between two objects:
 - Force $F = G (m_1 m_2) / D^2$
 - gravitational constant $G = 6.67 \times 10^{-11} \frac{Nm^2}{kg^2}$
 - m_i = mass of the two objects
 - D = distance between the two objects
- So both objects have same force applied to them
- $F = m * a \rightarrow a = F/m$
- On earth, assume mass of earth is so large it doesn't move, and D is constant:
 - Assume uniform acceleration
 - Position of falling object at time t :
 - $x(t) = x_0$
 - $y(t) = y_0 + 1/2 * 9.8 \text{ m/s}^2 * t^2$
 - Incrementally, $y \pm \text{gravity}$ (normalized to frame rate)
- Gravity Influences both bodies
- Can have two bodies orbit each other
- Only significant for large mass objects
- What happens after you apply a force to an object?
- What happens when you shoot a missile from a moving object?
- How to control a space ship?
- What about a game with planes?

Newtonian Physics (Mass)

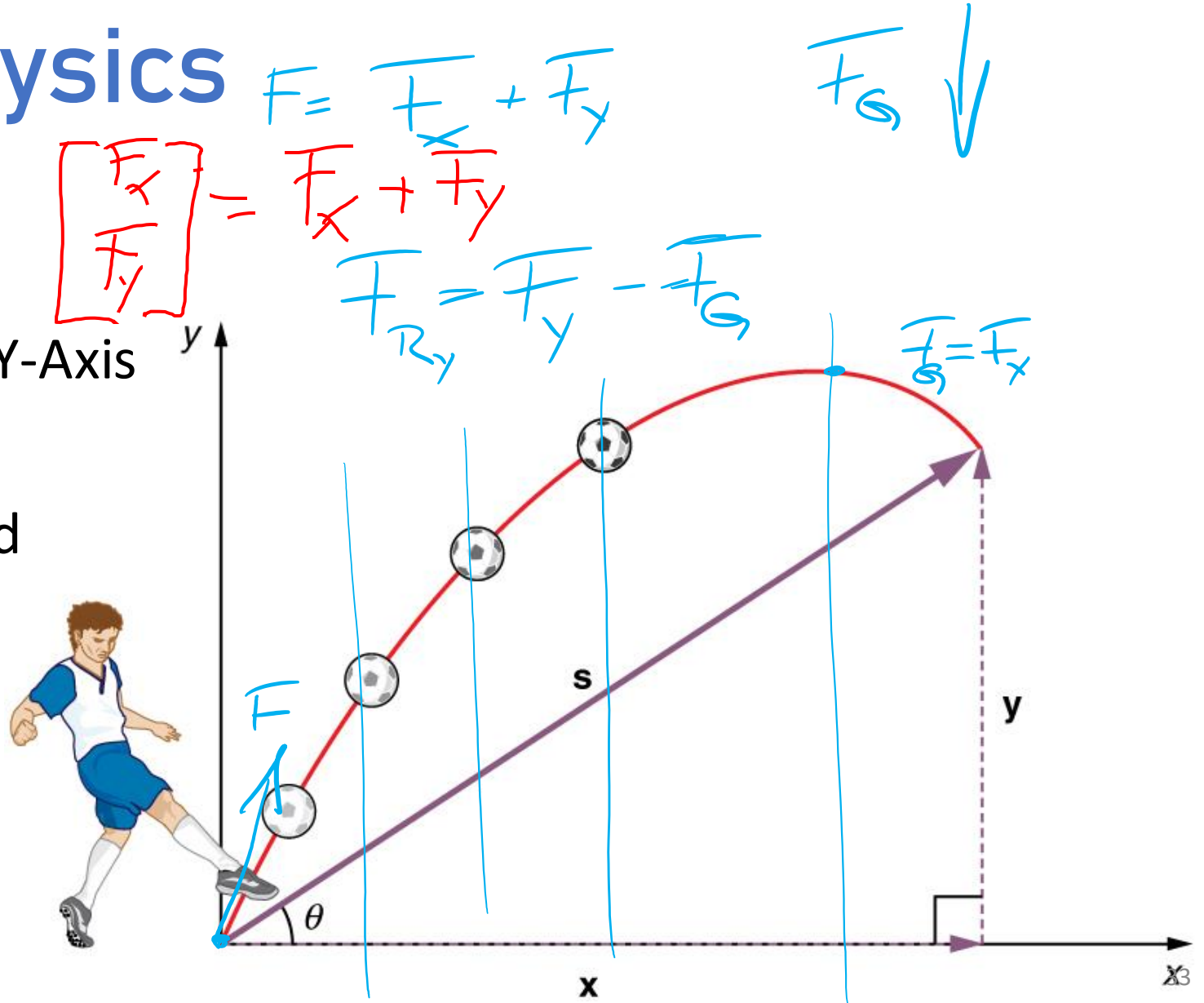
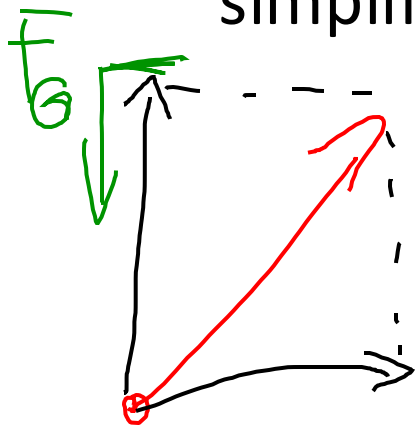
- Objects represented by their center of mass, not accurate for all physical effects
- Center of mass (x_c, y_c) for a polygon with n vertices:
 - Attach a mass to each vertex
 - $x_c = \frac{\sum_{i=0}^n x_i m_i}{\sum_{i=0}^n m_i}$
 - $y_c = \frac{\sum_{i=0}^n y_i m_i}{\sum_{i=0}^n m_i}$
- For sprites, put centre of mass where pixels are densest
- For arcade games, model gravity in sprite frames

$$\frac{x_0 + x_1}{2}$$



Newtonian Physics (Projectiles)

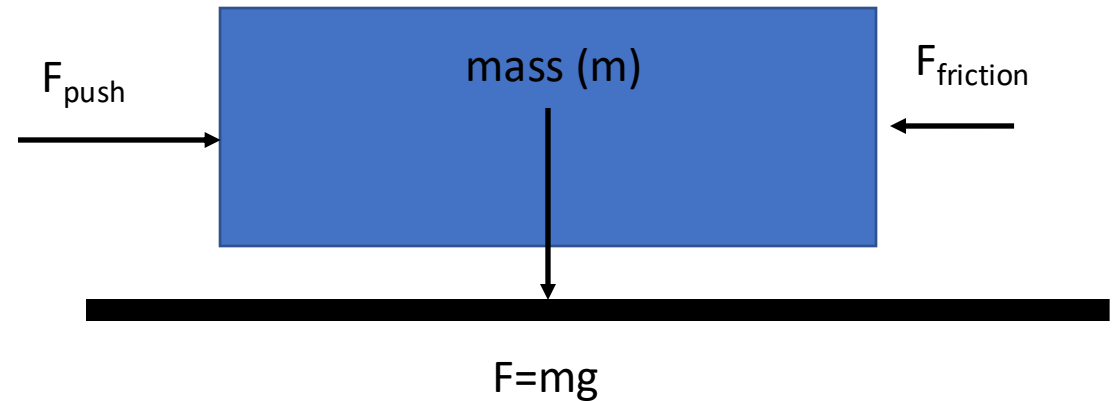
- Force applied at t_0
- Gravity applies on Y-Axis
- What about wind?
- for games: slice and simplify



Newtonian Physics

(Friction)

- Conversion of kinetic energy into heat
- Frictional force $F_{\text{friction}} = m g \mu$
 - m = mass, $g = 9.8 \text{ m/s}^2$,
 - μ = frictional coefficient = amount of force to maintain a constant speed
- $F_{\text{actual}} = F_{\text{push}} - F_{\text{friction}}$
- Careful that friction doesn't cause your object in the simulation to move backward!
- Consider inclined plane
- Usually two frictional forces:
 - Static friction when at rest (zero velocity). No movement unless overcome.
 - Kinetic friction when moving ($\mu_k < \mu_s$)



Newtonian Physics

(Friction)

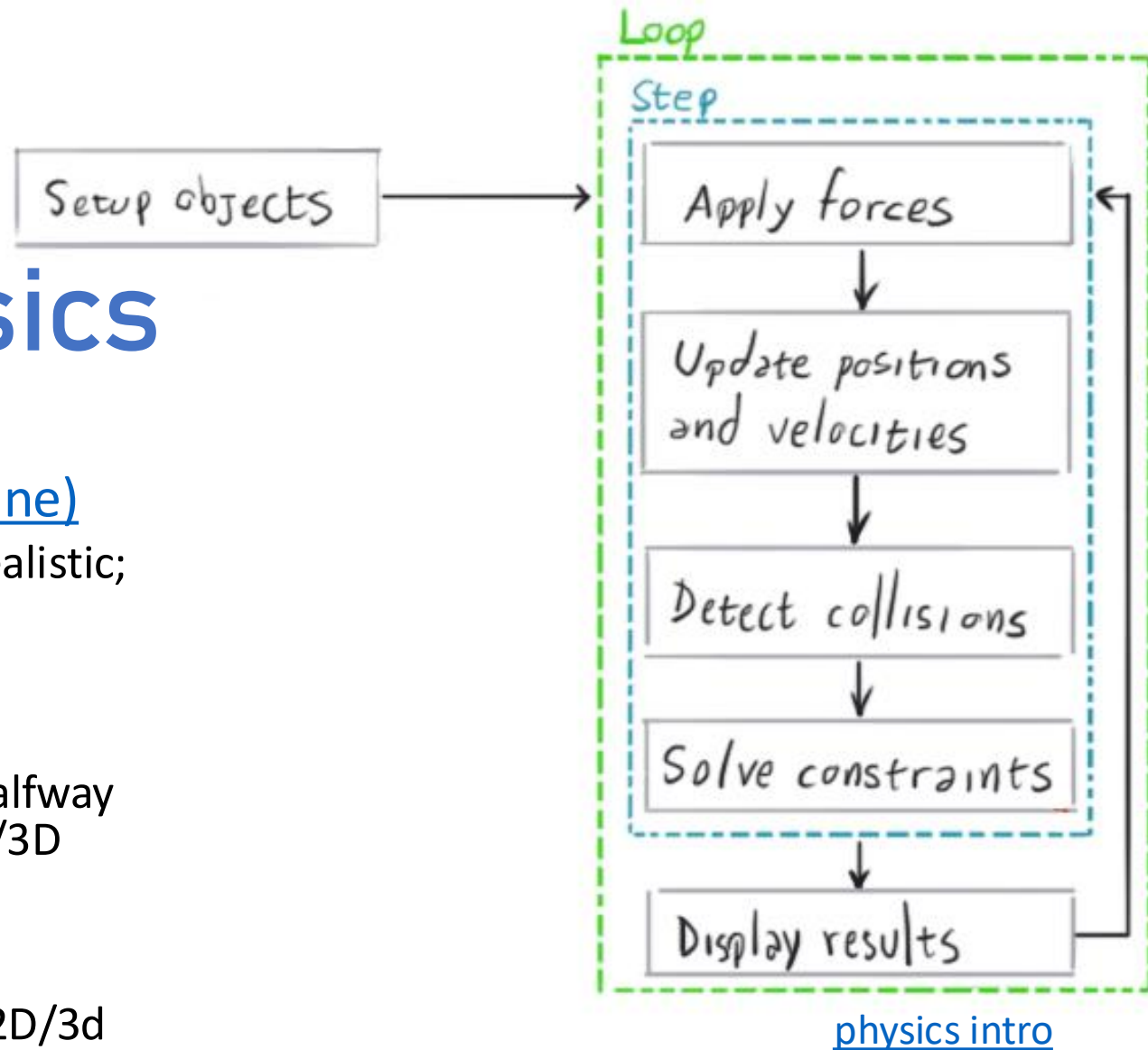
- Conversion of kinetic energy into heat
- Frictional force $F_{\text{friction}} = m g \mu$
 - m = mass, $g = 9.8 \text{ m/s}^2$,
 - μ = frictional coefficient = amount of force to maintain a constant speed
- $F_{\text{actual}} = F_{\text{push}} - F_{\text{friction}}$
- Careful that friction doesn't cause your object in the simulation to move backward!
- Consider inclined plane
- Usually two frictional forces:
 - Static friction when at rest (zero velocity). No movement unless overcome.
 - Kinetic friction when moving ($\mu_k < \mu_s$)

Surface Friction	Static (μ_s)	Kinetic (μ_k)
Steel on steel (dry)	0.6	0.4
Steel on steel (greasy)	0.1	0.05
Teflon on steel	0.041	0.04
Brake lining on cast iron	0.4	0.3
Rubber on concrete (dry)	1.0	0.9
Rubber on concrete (wet)	0.30	0.25
Metal on ice	0.022	0.02
Steel on steel	0.74	0.57
Aluminum on steel	0.61	0.47
Copper on steel	0.53	0.36
Nickel on nickel	1.1	0.53
Glass on glass	0.94	0.40
Copper on glass	0.68	0.53

Newtonian Physics

Physics Engines for Games:

- [ODE \(Open Dynamics Engine\)](#)
 - Open-Source; Good API; realistic; works well; not super performant
- [Bullet](#)
 - Open-Source; Good API; halfway between ODE & Physx; 2D/3D
- [PhysX \(Nvidia\)](#)
 - High performance; good support; games oriented; 2D/3d



Newtonian Physics

Physics Engines for Games:

- [ODE \(Open Dynamics Engine\)](#)
 - Open-Source; Good API; realistic; works well; not super performant
- [Bullet](#)
 - Open-Source; Good API; halfway between ODE & Physx; 2D/3D
- [PhysX \(Nvidia\)](#)
 - High performance; good support; games oriented; 2D/3d
- [Simple Physics](#)

What to know:

- needs to be treated like a separate part from visuals
- first use physics engine -> update visuals
- always use simplest shape

PCG meets Physics: TearDown



Optimisations for OpenGL

Topics for this lecture

- *Bundling Data*
- *Instancing Objects*
- *Normal Mapping*

Data Batches & Std::Types

- Unifying structures
- Reducing calls
- Reducing complexity

Data Batches & Std::Types

- Unifying structures
- Reducing calls
- Reducing Complexity

```
GLfloat vertices[] = {  
    -0.5f, -0.5f, -0.5f, //0 b l  
     0.5f, -0.5f, -0.5f, //1 b r  
     0.5f,  0.5f, -0.5f, //2 t r  
     0.5f,  0.5f, -0.5f, //3 t r  
    -0.5f,  0.5f, -0.5f, //4 t l  
    -0.5f, -0.5f, -0.5f, //5 b l  
    ...};
```

```
GLfloat normals[] {  
    0.0f,  0.0f, -1.0f,  
    0.0f,  0.0f, -1.0f,  
    0.0f,  0.0f, -1.0f,  
    0.0f,  0.0f, -1.0f,  
    0.0f,  0.0f, -1.0f,  
    0.0f,  0.0f, -1.0f,  
    ...};
```

```
GLuint indices[][3] = {  
    {0, 1, 2}, // first Triangle front  
    {3, 4, 5}, // second Triangle  
    ...  
    {30, 31, 32 }, // first Triangle back  
    {33, 34, 35 }  // second Triangle  
};
```

```
GLfloat colours[][4] = {  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    ...};  
GLfloat texture_coords[] = {  
    0.0f, 0.0f,  
    1.0f, 0.0f,  
    1.0f, 1.0f,  
    1.0f, 1.0f,  
    0.0f, 1.0f,  
    0.0f, 0.0f,  
    ...};
```


Data Batches & Std::Types

- Unifying structures
- Reducing calls
- Reducing Complexity

```
std::vector<GLfloat> vertices = {  
    -0.5f, -0.5f, -0.5f, //0 b l  
    0.5f, -0.5f, -0.5f, //1 b r  
    0.5f, 0.5f, -0.5f, //2 t r  
    0.5f, 0.5f, -0.5f, //3 t l  
    -0.5f, 0.5f, -0.5f, //4 t l  
    -0.5f, -0.5f, -0.5f, //5 b l  
...};
```

```
std::vector<GLfloat> normals{  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
...};
```

```
std::vector<GLuint> indices[] = {  
    0, 1, 2, // first Triangle front  
    3, 4, 5, // second Triangle  
    ...  
    30, 31, 32, // first Triangle back  
    33, 34, 35 // second Triangle  
};
```

```
GLfloat colours[][4] = {  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    ...};  
GLfloat texture_coords[] = {  
    0.0f, 0.0f,  
    1.0f, 0.0f,  
    1.0f, 1.0f,  
    1.0f, 1.0f,  
    0.0f, 1.0f,  
    0.0f, 0.0f,  
    ...};
```

Data Batches & Std::Types

- Unifying structures
- Reducing calls
- Reducing Complexity

```
std::vector<GLfloat> vertices = {  
    -0.5f, -0.5f, -0.5f, //0 b l  
    0.5f, -0.5f, -0.5f, //1 b r  
    0.5f, 0.5f, -0.5f, //2 t r  
    0.5f, 0.5f, -0.5f, //3 t r  
    -0.5f, 0.5f, -0.5f, //4 t l  
    -0.5f, -0.5f, -0.5f, //5 b l  
    ...};  
  
std::vector<GLfloat> normals{  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    0.0f, 0.0f, -1.0f,  
    ...};  
  
std::vector<GLuint> indices[] = {  
    0, 1, 2, // first Triangle front  
    3, 4, 5, // second Triangle  
    ...  
    30, 31, 32, // first Triangle back  
    33, 34, 35 // second Triangle  
};
```

```
GLfloat colours[][4] = {  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    { 1.0f, 0.0f, 0.0f, 1.0f },  
    ...};  
  
GLfloat texture_coords[] = {  
    0.0f, 0.0f,  
    1.0f, 0.0f,  
    1.0f, 1.0f,  
    1.0f, 1.0f,  
    0.0f, 1.0f,  
    0.0f, 0.0f,  
    ...};
```

```
glGenBuffers(NumBuffers, Buffers);
```

```
glBindBuffer(GL_ARRAY_BUFFER, Buffers[Triangles]);  
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), vertices.data(),  
GL_STATIC_DRAW);
```

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, Buffers[Indices]);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLfloat),  
indices.data(), GL_STATIC_DRAW);
```

Data Batches & Std::Types

- Unifying structures
- Reducing calls
- Reducing Complexity

```
std::vector<GLfloat> vertices = {  
    -0.5f, -0.5f, 0.0f, // b l  
    0.5f, -0.5f, 0.0f, // b r  
    0.5f, 0.5f, 0.0f, // t r  
    0.5f, 0.5f, -1.0f, // t b r  
    -0.5f, 0.5f, -1.0f, // t b l  
    -0.5f, 0.5f, 0.0f, // l  
    ...  
};  
  
std::vector<GLfloat> colours[4] = {  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    {0.0f, 0.0f, -1.0f, 1.0f},  
    ...  
};
```

```
std::vector<GLuint> indices[] = {  
    0, 1, 2, // first Triangle front  
    3, 4, 5, // second Triangle  
    ...  
    30, 31, 32, // first Triangle back  
    33, 34, 35 // second Triangle  
};  
  
std::vector<GLfloat> colours[4] = {  
    {0.0f, 0.0f, 1.0f, 1.0f},  
    {0.0f, 0.0f, 1.0f, 1.0f},  
    {0.0f, 0.0f, 1.0f, 1.0f},  
    {0.0f, 0.0f, 1.0f, 1.0f},  
    ...  
};
```

```
glGenBuffers(NumBuffers, Buffers);  
  
glBindBuffer(GL_ARRAY_BUFFER, Buffers[Triangles]);  
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), vertices.data(),  
             GL_STATIC_DRAW);  
  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, Buffers[Indices]);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLfloat),  
             indices.data(), GL_STATIC_DRAW);
```

Data Batches

- Benefits

- Faster load times
- No indexing issues
- Great for cohesion of data

- Costs

- More difficult to debug
- Not great for data swapping
- Does not work well with extremely large data sets
- Load time for large bundles

```
std::vector<GLfloat> vertices = {  
//0  x,   y,   z,   s,   t,   nx, ny,  nz,  r,   g,  b,   a  
    -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
      0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
      0.5f,  0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
      0.5f,  0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
     -0.5f,  0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
     -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f  
};
```

Data Batches

- Benefits
 - Faster load times
 - No indexing issues
 - Great for cohesion of data

- Costs

- More difficult to debug
- Not great for data swapping
- Does not work well with extremely large data sets
- Load time for large bundles

```
std::vector<GLfloat> vertices = {  
//0  x,   y,   z,   s,   t,  nx, ny,  nz,  r,   g,   b,   a  
-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
 0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
};
```

```
GLuint Buffers[1];  
glGenBuffers(1, Buffers);  
  
glBindBuffer(GL_ARRAY_BUFFER, Buffers[VERTICES]);  
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(GLfloat), vertices.data(), GL_STATIC_DRAW);  
  
glVertexAttribPointer(VERTICES, 3, GL_FLOAT,  
GL_FALSE, 12 * sizeof(GLfloat), BUFFER_OFFSET(0));  
glVertexAttribPointer(Textures, 2, GL_FLOAT,  
GL_FALSE, 12 * sizeof(GLfloat), BUFFER_OFFSET(3 * sizeof(GLfloat)));  
glVertexAttribPointer(Normals, 3, GL_FLOAT,  
GL_FALSE, 12 * sizeof(GLfloat), BUFFER_OFFSET(5 * sizeof(GLfloat)));  
glVertexAttribPointer(Colours, 4, GL_FLOAT,  
GL_FALSE, 12 * sizeof(GLfloat), BUFFER_OFFSET(8 * sizeof(GLfloat)));
```

Instancing

- Benefits

- Faster load times
- Memory efficient
- Scalable

- Costs

- More difficult to setup
- Does not work well with custom changes

```
for (int i=0;i<objects.size();i++)
{
    glBindVertexArray(VAOs[i]);
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    int mvLoc = glGetUniformLocation(shader, "model");
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE,
        glm::value_ptr(models[i]));
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
}
```

Instancing

```
int instances = 1000;
for (unsigned int i = 0; i < instances; i++)
{
    std::stringstream ss;
    std::string index;
    ss << i;
    index = ss.str();
    int oLoc = glGetUniformLocation(shader, ("offsets[" + index + "]").c_str());
    glUniform2fv(oLoc, 1, glm::value_ptr(translations[i]));
}

for (int i=0;i<objects.size();i++)
{
    glBindVertexArray(VAOs[i]);
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    int mvLoc = glGetUniformLocation(shader, "model");
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(models[i]));
    glDrawElementsInstanced(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0, instances);
}
```

Immersive Systems Technologies

- Benefits
 - Faster load times
 - Memory efficient
 - Scalable
- Costs
 - More difficult to setup
 - Does not work well with custom changes

```
#version 400 core

layout( location = 0 ) in vec3 vPosition;
layout( location = 1 ) in vec4 vColour;
layout (location = 2) in vec2 aTexCoord;
uniform mat4 mvp;

uniform vec2
uniform vec3 offsets[100];

out vec4 fragColour;
out vec2 TexCoord;

void
main()
{
    vec3 offset = offsets[gl_InstanceID];
    gl_Position = mvp * vec4 (vPosition+offset, 1.0);
    fragColour = vec4(0.5, 0.4, 0.8, 1.0);
    fragColour = vColour;
    TexCoord = aTexCoord;
}
```

Instancing

- Benefits
 - Faster load times
 - Memory efficient
 - Scalable
- Costs
 - More difficult to setup
 - Does not work well with custom changes

```
int instances = 1000;
for (unsigned int i = 0; i < instances; i++)
{
    std::stringstream ss;
    std::string index;
    ss << i;
    index = ss.str();
    int oLoc = glGetUniformLocation(shader, ("offsets[" + index + "]").c_str());
    glUniform2fv(oLoc, 1, glm::value_ptr(translations[i]));
}

for (int i=0;i<objects.size();i++)
{
    glBindVertexArray(VAOs[i]);
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    int mvLoc = glGetUniformLocation(shader, "model");
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(models[i]));
    glDrawElementsInstanced(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0, instances);
}
```

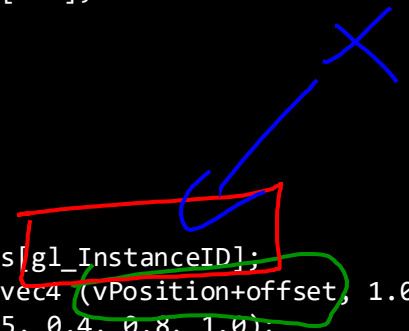
```
#version 400 core

layout( location = 0 ) in vec3 vPosition;
layout( location = 1 ) in vec4 vColour;
layout( location = 2 ) in vec2 aTexCoord;
uniform mat4 mvp;

uniform vec3 offsets[100];

out vec4 fragColour;
out vec2 TexCoord;

void
main()
{
    vec3 offset = offsets[gl_InstanceID];
    gl_Position = mvp * vec4(vPosition+offset, 1.0);
    fragColour = vec4(0.5, 0.4, 0.8, 1.0);
    fragColour = vColour;
    TexCoord = aTexCoord;
}
```



Instancing

- Benefits
 - Faster load times
 - Memory efficient
 - Scalable

```
int instances = 1000;
for (unsigned int i = 0; i < instances; i++)
{
    std::stringstream ss;
    std::string index;
    ss << i;
    index = ss.str();
    int oLoc = glGetUniformLocation(shader, ("offsets[" + index + "]").c_str());
    glUniform2fv(oLoc, 1, glm::value_ptr(translations[i]));
}

for (int i=0;i<objects.size();i++)
{
    glBindVertexArray(VAOs[i]);
    glBindTexture(GL_TEXTURE_2D, textures[i]);
    int mvLoc = glGetUniformLocation(shader, "model");
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(models[i]));
    glDrawElementsInstanced(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0, instances);
}
```

- Costs

- More difficult to setup
- Does not work well with custom

Next step: Remove uniform and switch to
`glVertexAttribDivisor(&location, #instance);`

```
#version 400 core

in vec3 vPosition;
in vec4 vColour;
in vec2 aTexCoord;

out vec4 outColour;

void
main()
{
    vec3 offset = offsets[gl_InstanceID];
    gl_Position = mvp * vec4 (vPosition+offset, 1.0);
    fragColour = vec4(0.5, 0.4, 0.8, 1.0);
    fragColour = vColour;
    TexCoord = aTexCoord;
}
```

Instancing

```
glBindBuffer(GL_ARRAY_BUFFER, Buffers[Offsets]);  
glBufferStorage(GL_ARRAY_BUFFER, offsets.size() * sizeof(GLfloat) , &offsets[0], 0);
```

```
glVertexAttribPointer(Offsets, 3, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));  
glBindBuffer(GL_ARRAY_BUFFER, Offsets);  
glVertexAttribDivisor(Offsets, 1);
```

```
...  
Void display(){  
...  
glDrawElementsInstanced(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0, instances);  
};
```

- Benefits
 - Faster load times
 - Memory efficient
 - Scalable

- Costs
 - More difficult to setup
 - Does not work well with custom changes

Further Reading:

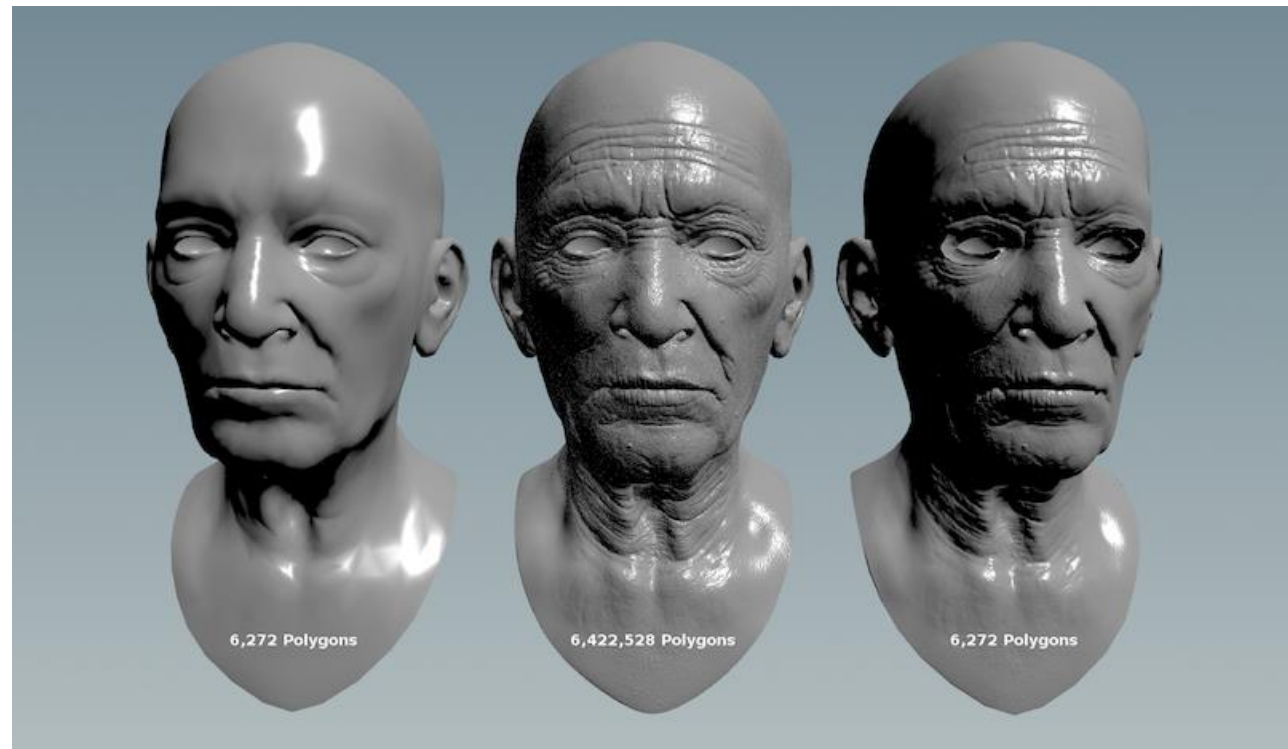
- opengl-tutorial.org
- learnopengl.com

```
#version 400 core  
  
layout( location = 0 ) in vec3 vPosition;  
layout( location = 1 ) in vec4 vColour;  
layout (location = 2) in vec2 aTexCoord;  
layout (location = 3) in vec3 aOffset;  
  
out vec4 fragColour;  
out vec2 TexCoord;  
  
void  
main()  
{  
    gl_Position = mvp * vec4 (vPosition+aOffset, 1.0);  
    fragColour = vec4(0.5, 0.4, 0.8, 1.0);  
    fragColour = vColour;  
    TexCoord = aTexCoord;  
}
```

Normal Mapping

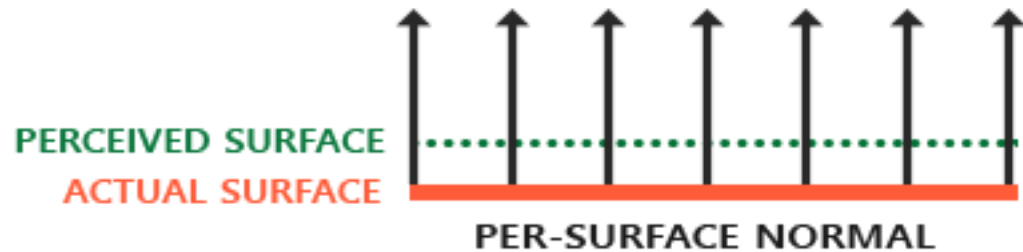
- Benefits
 - Reduces need for complex geometry
 - Allows for quick geometry variations

- Costs
 - Runs in the Shader
 - Can be spotted

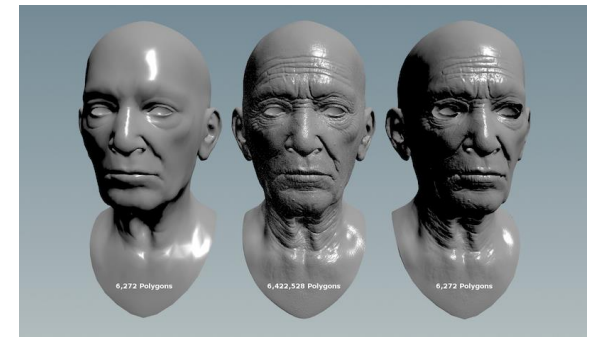
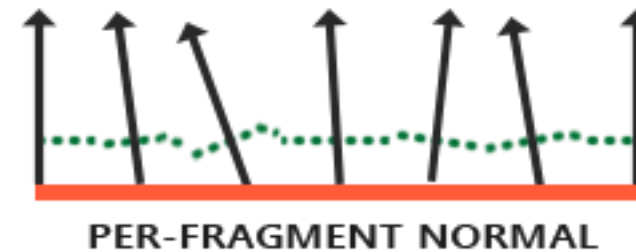


Normal Mapping

- Benefits
 - Reduces need for complex geometry
 - Allows for quick geometry variations

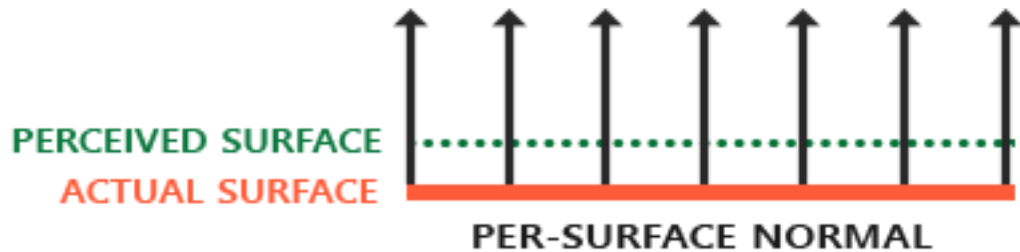


- Costs
 - Runs in the Shader
 - Can be spotted

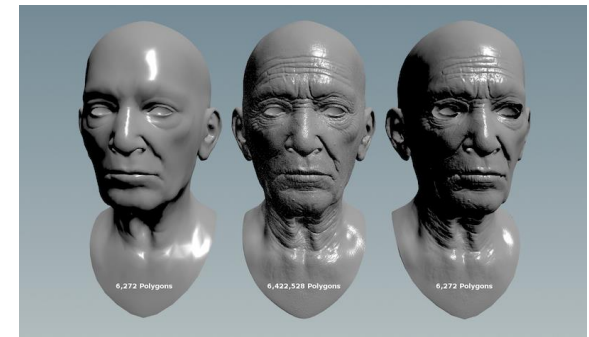
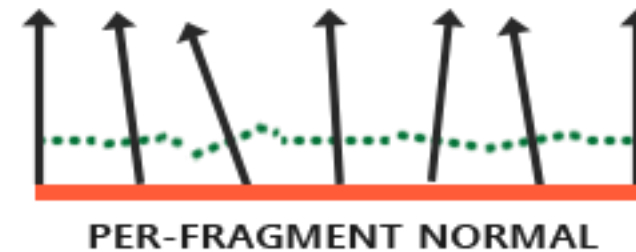


Normal Mapping

- Benefits
 - Reduces need for complex geometry
 - Allows for quick geometry variations

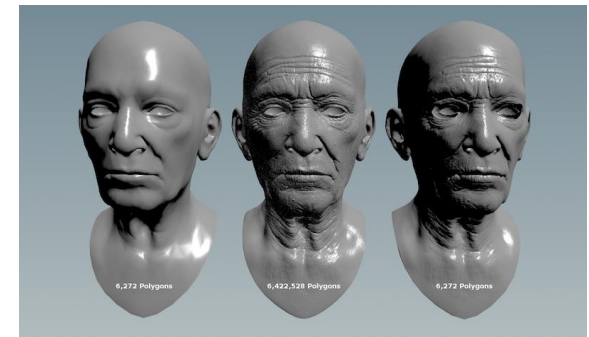


- Costs
 - Runs in the Shader
 - Can be spotted



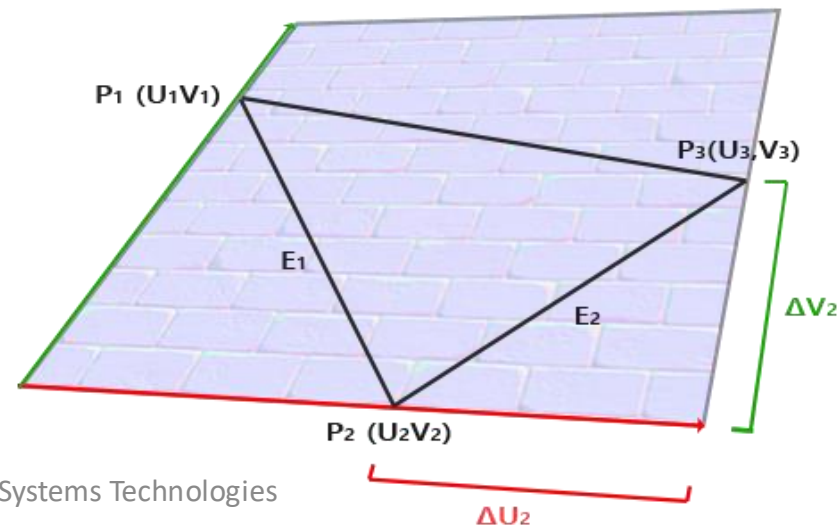
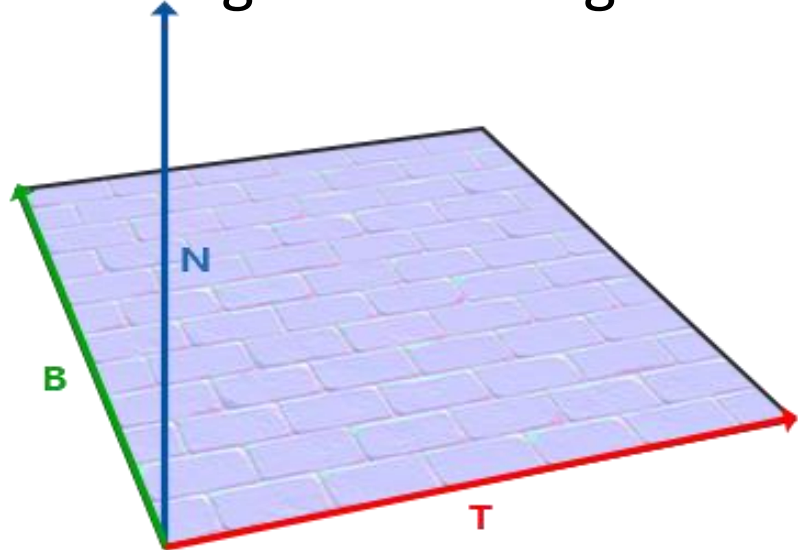
Normal Mapping

- Blue tint due to normal are pointing closely towards Y – Axis
- To correctly use them need tangents and bi-tangents for triangles



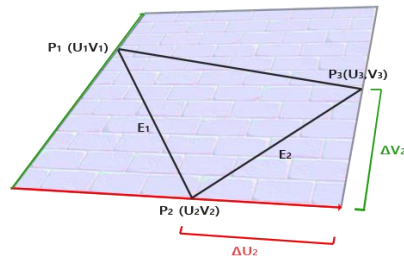
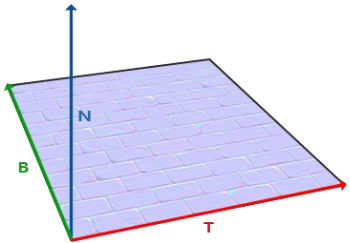
Normal Mapping

- Blue tint due to normal are pointing closely towards Y – Axis
- To correctly use them need tangents and bi-tangents for triangles



Normal Mapping

- Blue tint due to normal are pointing closely towards Y – Axis
- To correctly use them need tangents and bitangents for triangles



```
std::vector<glm::vec3> verts = {
    //0 x, y, z,
    glm::vec3(-0.5f, -0.5f, -0.5f),
    glm::vec3(0.5f, -0.5f, -0.5f),
    glm::vec3(0.5f, 0.5f, -0.5f),
    glm::vec3(0.5f, 0.5f, -0.5f),
    glm::vec3(-0.5f, 0.5f, -0.5f),
    glm::vec3(-0.5f, -0.5f, -0.5f),
};
```

```
std::vector<glm::vec2> texs = {
    //0 s, t,
    glm::vec2(0.0f, 0.0f),
    glm::vec2(1.0f, 0.0f),
    glm::vec2(1.0f, 1.0f),
    glm::vec2(1.0f, 1.0f),
    glm::vec2(0.0f, 1.0f),
    glm::vec2(0.0f, 0.0f),
};
```

```
for (int f = 0; f < verts.size() - 3; f += 3) {
```

```
    glm::vec3 edge1 = verts[f + 1] - verts[f];
    glm::vec3 edge2 = verts[f + 2] - verts[f];
    glm::vec2 deltaUV1 = texs[f + 1] - texs[f];
    glm::vec2 deltaUV2 = texs[f + 2] - texs[f];
    glm::vec3 tangent;
    glm::vec3 bitangent;
```

```
    float f1 = 1.0f / (deltaUV1.x * deltaUV2.y - deltaUV2.x * deltaUV1.y);
```

```
    tangent.x = f1 * (deltaUV2.y * edge1.x - deltaUV1.y * edge2.x);
    tangent.y = f1 * (deltaUV2.y * edge1.y - deltaUV1.y * edge2.y);
    tangent.z = f1 * (deltaUV2.y * edge1.z - deltaUV1.y * edge2.z);
    tangent = glm::normalize(tangent);
```

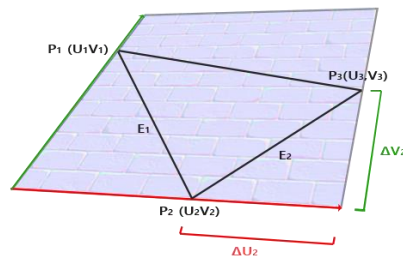
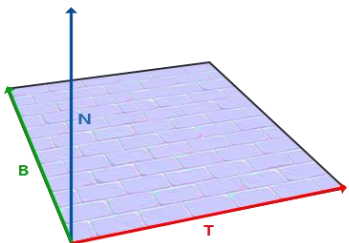
```
    bitangent.x = f1 * (-deltaUV2.x * edge1.x + deltaUV1.x * edge2.x);
    bitangent.y = f1 * (-deltaUV2.x * edge1.y + deltaUV1.x * edge2.y);
    bitangent.z = f1 * (-deltaUV2.x * edge1.z + deltaUV1.x * edge2.z);
    bitangent = glm::normalize(bitangent);
```





Normal Mapping

- Blue tint due to normal are pointing closely towards Y – Axis
- To correctly use them need tangents and bi-tangents for triangles



```
#version 450 core

in vec4 fragColour;
in vec2 TexCoord;
in mat3 TBN;
in vec3 fragPos;

uniform sampler2D ourTexture;
uniform sampler2D normalMap;
uniform vec3 lightPos;
uniform vec3 viewPos;

out vec4 fColor;

void main()
{
    fColor = texture(ourTexture, TexCoord) * fragColour;

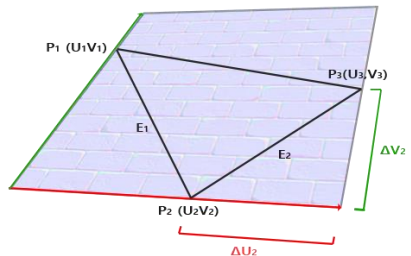
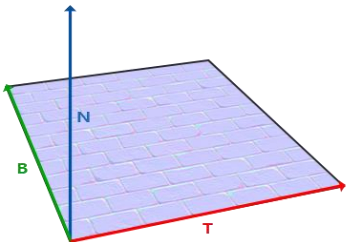
    vec3 normal = texture(normalMap, TexCoord).rgb;
    normal = normalize(normal * 2.0 - 1.0);

    vec3 lightDir = TBN * normalize(lightPos - fragPos);
    vec3 viewDir = TBN * normalize(viewPos - fragPos);

    ...
}
```

Normal Mapping

- Blue tint due to normal are pointing closely towards Y – Axis
- To correctly use them need tangents and bi-tangents for triangles



Thanks so far!

Any Questions?