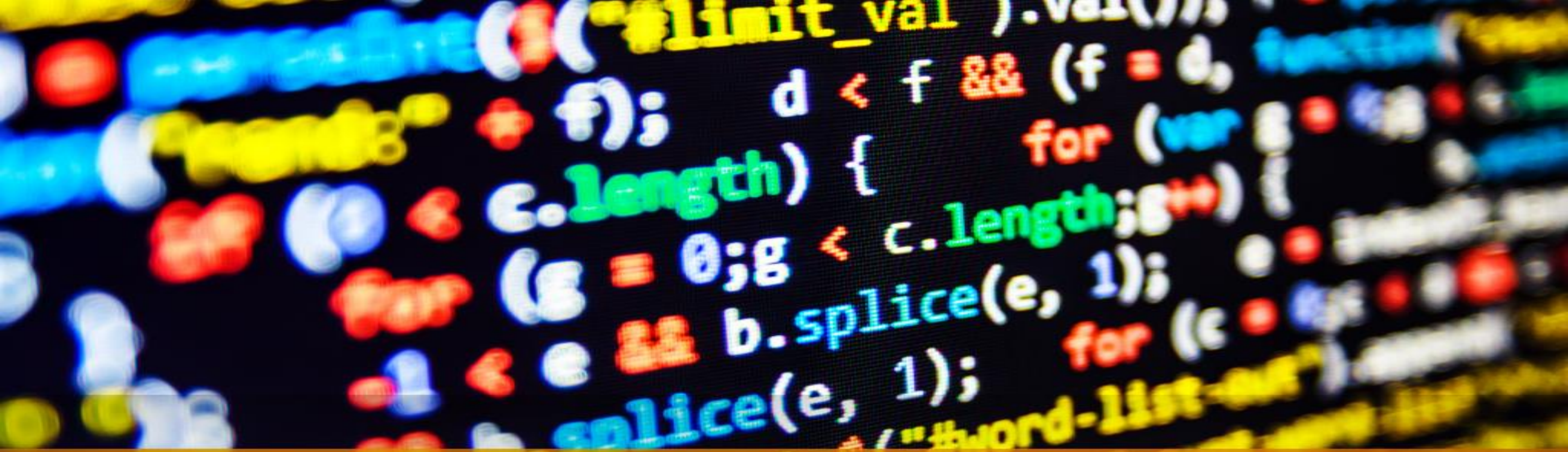


Lecture 09

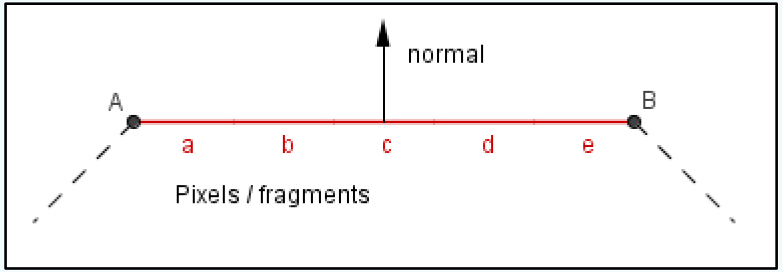
Lighting, Shading, & Texture Mapping (Part 2)

Prepared by Ban Kar Weng (William)

Shading

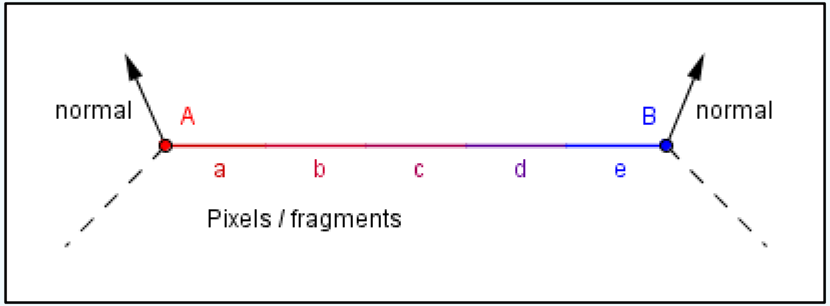
Shading

Shading | Flat Shading



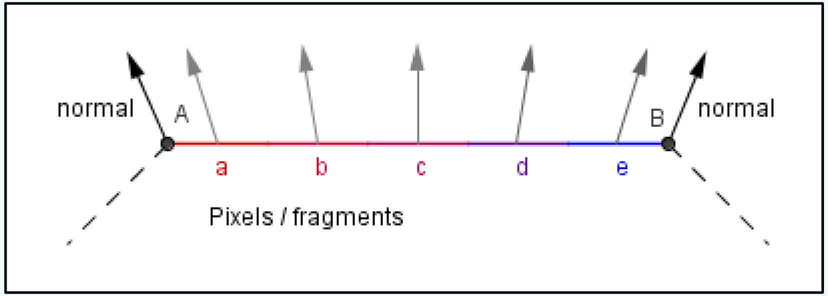
- **Per-polygon** shading
- Use one surface normal per polygon.
- The colour is uniform on that polygon.

Shading | Gouraud Shading



- **Per-vertex** shading
- Use one normal per vertex
- The colour is interpolated over the polygon.
- Drawback: specular highlights that occurs inside the polygon would not be shown.

Shading | Gouraud Shading

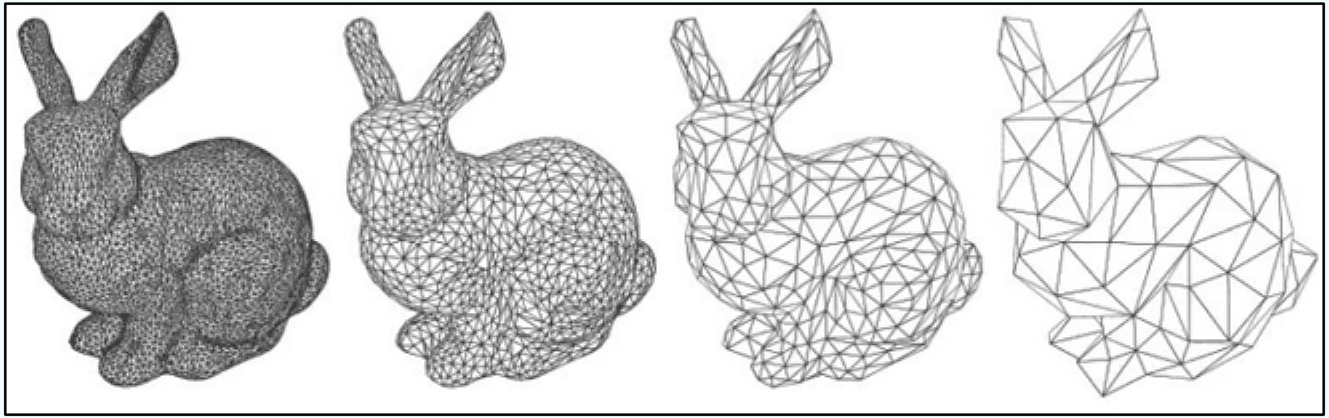


- **Per-fragment** shading
- Use one normal per vertex
- But normal is interpolated across vertices.
- Colour is computed using the interpolated normal.
- Accounts for specular highlights inside the polygon.

Texturing

Texturing | Motivation

To improve realism, we can add more vertices and therefore specify more colours.

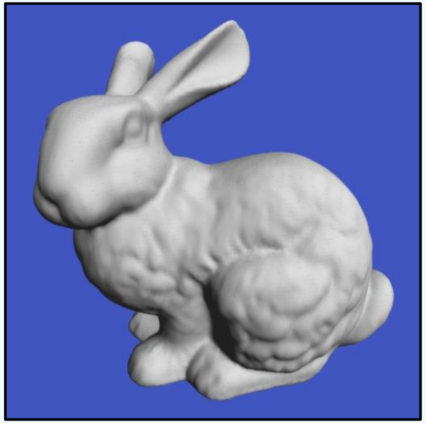


However, this incurs memory overhead.

Texturing | Motivation

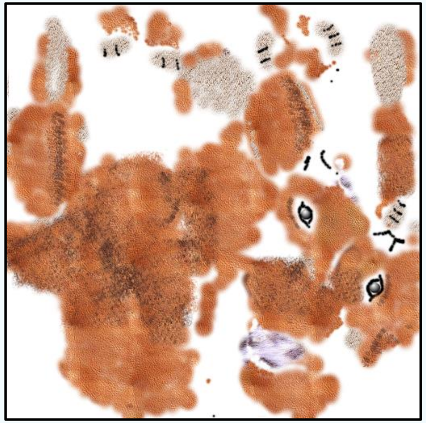
Texturing is a technique for efficiently modelling variations in surface's material and finish.

Instead of precisely representing the geometry of the object, a texture image is applied to the object.



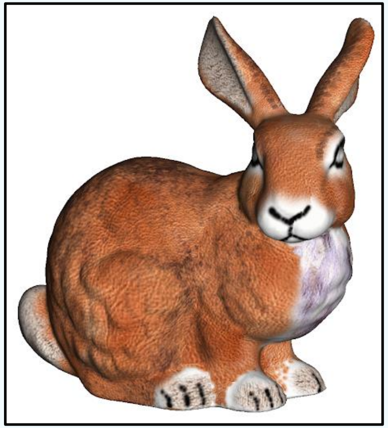
Object

+



Texture image

=

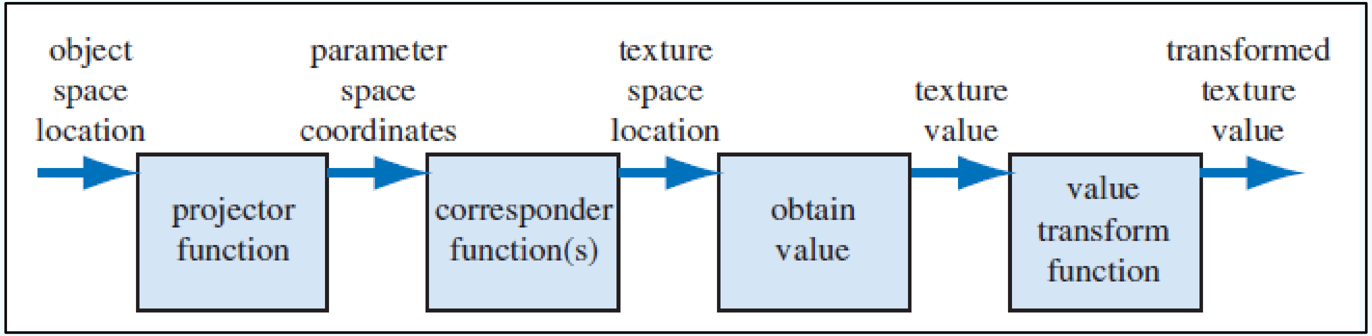


Textured Object

All images are taken online. To get the link, right click the image and select "Edit Hyperlink" in the context menu.

Texturing | The Texturing Pipeline

The generalized texturing pipeline for a single texture image.



- **Input**: Location of a surface point in object space.
- **Output**: Transformed texture value (e.g. colour from the texture image).

NOTE:
This pipeline covers all texturing use cases, hence its complexity.
For basic use case, **not all steps** are required at all times, as we shall see.

Image taken from Figure 6.2 Real-time Rendering (4th Edition)

Texturing | The Texturing Pipeline

Object space location

- The location of a point on the surface to be textured in model space.

Parameter space coordinates

- More commonly known as ***texture coordinates***.
- 2D texture coordinates is commonly referred to as (u, v) or (s, t).
- Has a $[\mathbf{0}, \mathbf{1}]$ range.
- $(0,0)$ is for the **lower left corner** of the texture image (in OpenGL).
- $(1,1)$ is for the **upper right corner** of the texture image (in OpenGL).

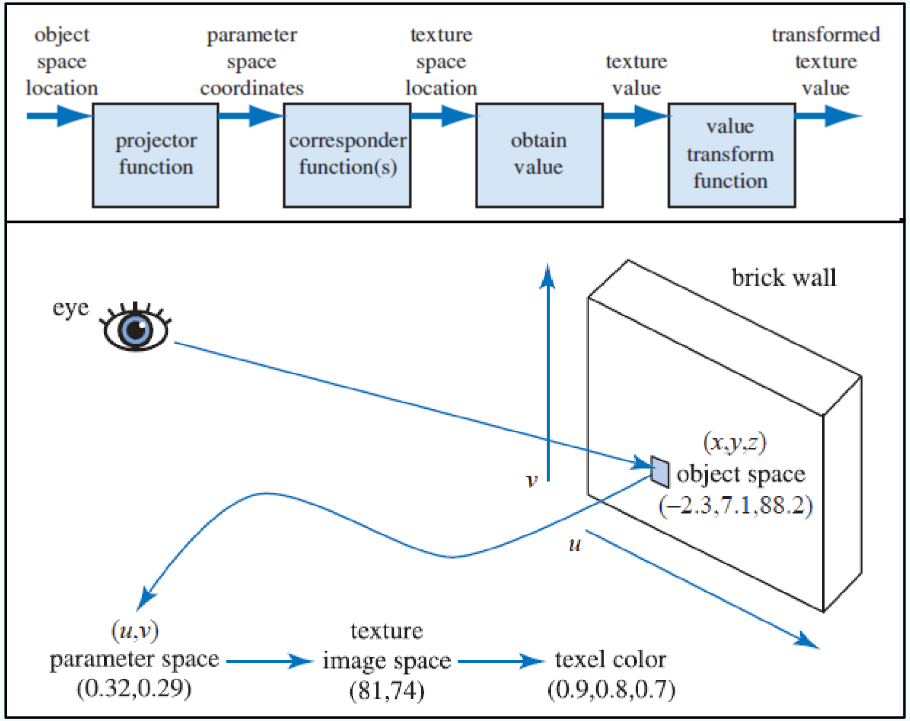


Image taken from Figure 6.2 Real-time Rendering (4th Edition)

Texturing | The Texturing Pipeline

Texture space location

- Refers to location of the pixel in the texture image. ***texel*** (*short for texture element*) in the texture image.
- Typically, the word ***texel*** (*short for texture element*) is used to refer to pixel in the texture image, to differentiate them from the pixels on the screen.

Texture value

- The value stored a texel.

Transformed texture value

- A new value obtained by transforming the texture value.

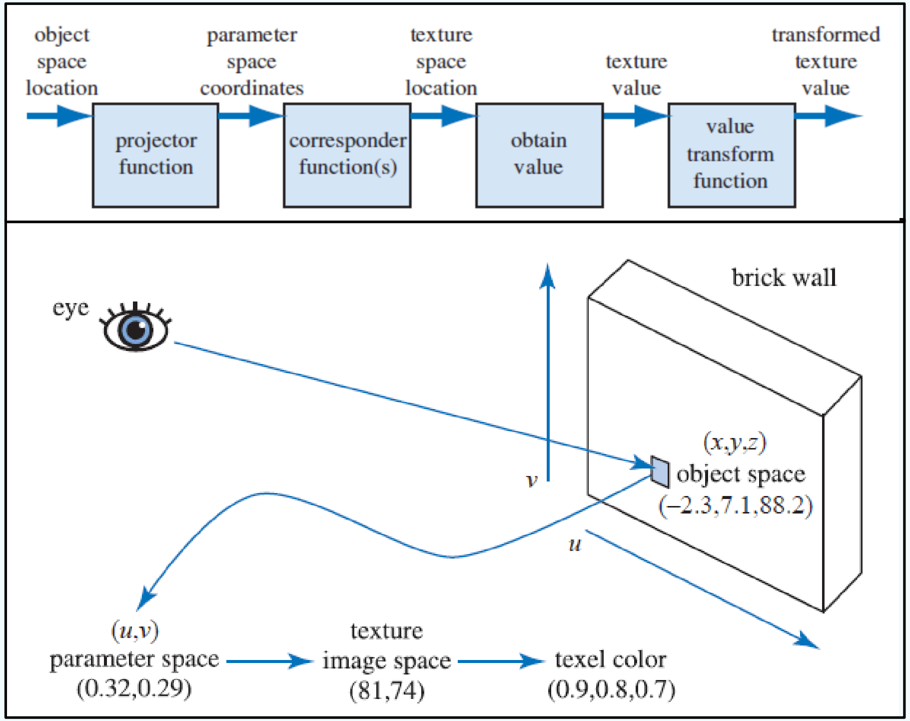


Image taken from Figure 6.2 Real-time Rendering (4th Edition)

Texturing | The Texturing Pipeline

Example: Texturing a brick wall texture image on a triangle.

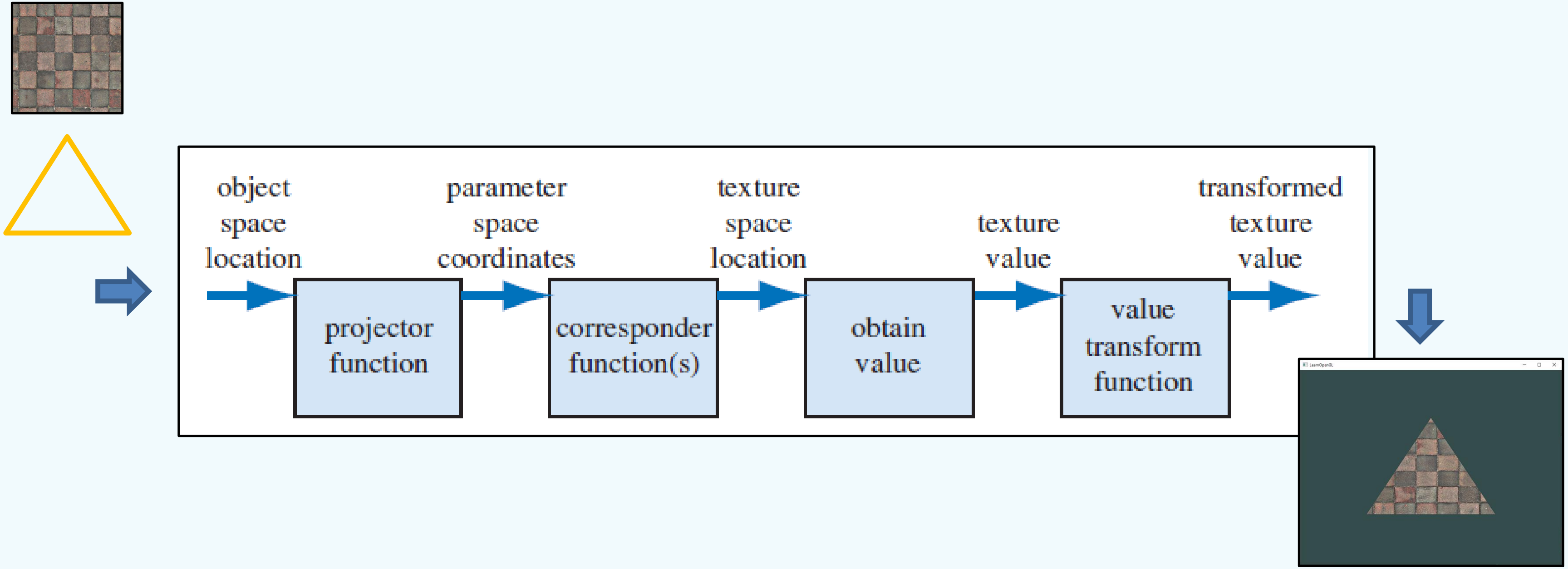
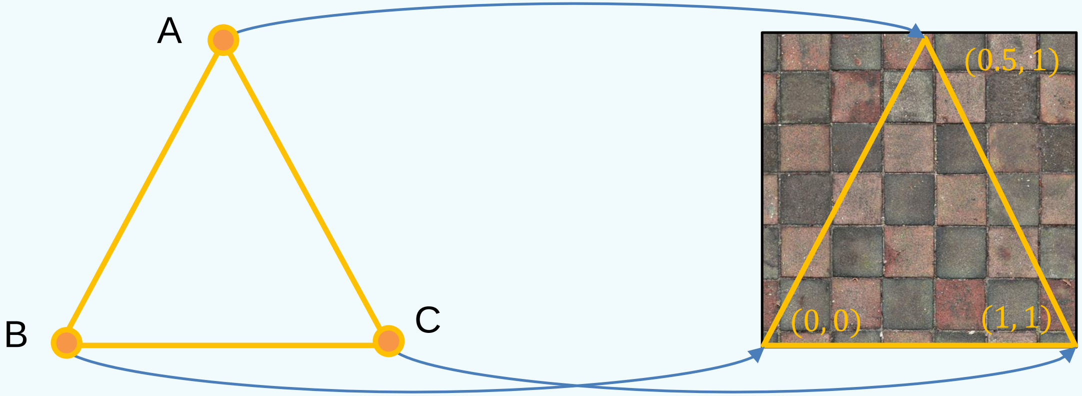


Image taken from Figure 6.2 Real-time Rendering (4th Edition)

Texturing | The Texturing Pipeline

Example: Texturing a brick wall texture image on a triangle.



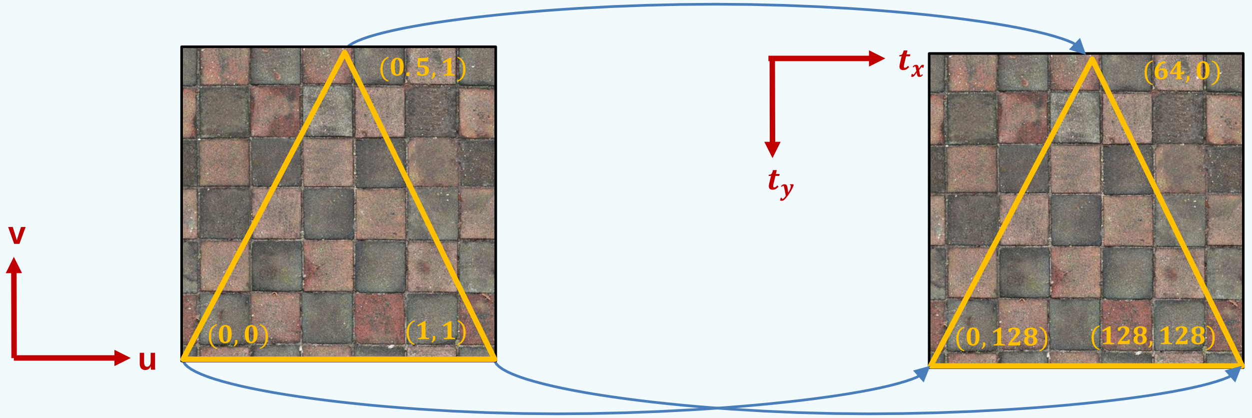
Projector function

- Associate an object space location with parameter space coordinates (or ***texture coordinates***)
- In this example, vertices A, B, and C are associated to $(0.5,1)$, $(0,0)$, and $(1,1)$ respectively.

NOTE: Explicit association is done on vertices only.
Fragments within the triangle get their texture coordinates through interpolation in the rasterization stage.

Texturing | The Texturing Pipeline

Example: Texturing a brick wall texture image on a triangle.

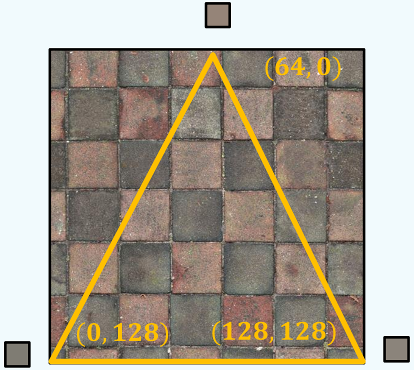


Corresponder function

- Convert the texture coordinate into texture space location (in pixel unit).
- In this example, if size of texture image is 128 x 128, then texture coordinate $(0.5,1)$ is converted to texture space location $(64, 0)$.

Texturing | The Texturing Pipeline

Example: Texturing a brick wall texture image on a triangle.



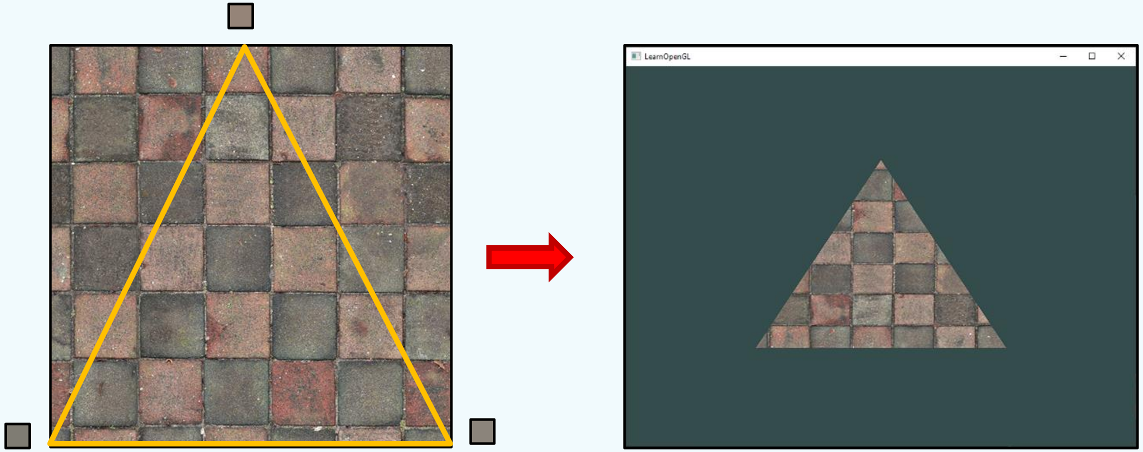
Obtain Value

- Get the texture value at the given texture space location.
- In this example, the texture value is simply the colour.

NOTE: To facilitate explanation, we show the texture value of the vertices only. In reality, texture value lookup is also performed on fragments within the triangle.

Texturing | The Texturing Pipeline

Example: Texturing a brick wall texture image on a triangle.



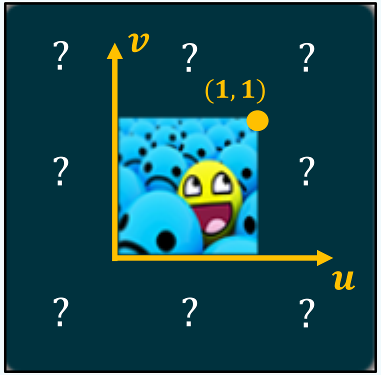
Value Transform Function

- Transform the texture value to get new value used to modify some property of the surface (e.g. colour conversion from sRGB to RGB)
- In this example, there is no value transform function as the texture value is used as it is.

Image taken from Figure 6.2 Real-time Rendering (4th Edition)

Texture Wrapping

Texture Mapping | Texture Wrapping



- An image will appear on the surface where texture coordinate (u, v) are in the $[0,1]$ range.
- But what happens outside this range?

Texture Mapping | Texture Wrapping


GL_REPEAT


GL_MIRRORED_REPEAT


GL_CLAMP_TO_EDGE

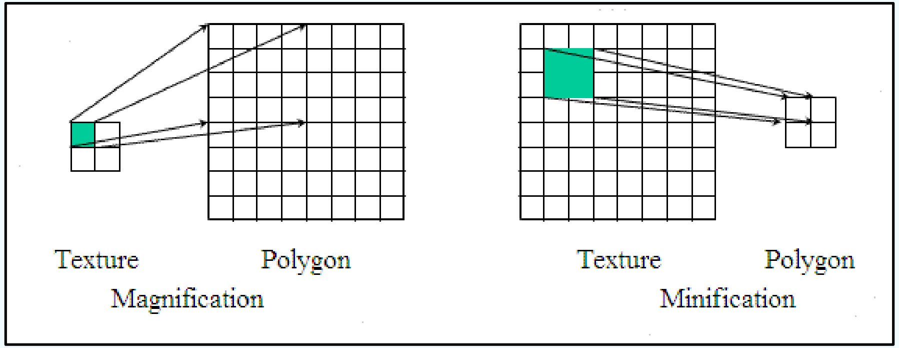

GL_CLAMP_TO_BORDER

OpenGL handles it with **texture wrapping** mode:

- ***Repeat*** – the image repeats itself across the surface.
- ***Mirror*** – similar to the Repeat mode, except the image is flipped on every other repetition.
- ***Clamp to Edge*** - values outside the range $[0,1]$ are clamped to this range.
- ***Clamp to Border*** – texture coordinates outside $[0,1]$ are rendered with a separately defined border colour.

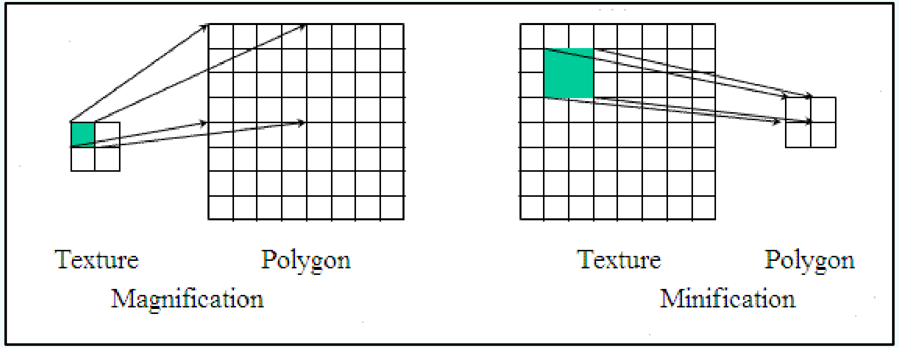
Texture Filtering

Texture Filtering | Motivation



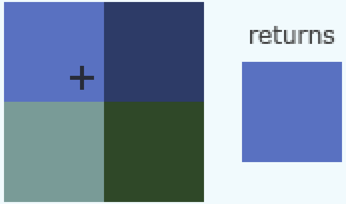
- The number of screen pixels covered by a surface **could differs significantly** from the number of texels applied on the surface.
- Possible cases:
  - **Magnification** (texels < pixels) – texture must be **magnified** to cover the pixels.
  - **Minification** (texels > pixels) – texture must be **minimized** to cover the pixels.
- Both cases would lead to aliasing in the final rendered image.

Texture Filtering | Motivation



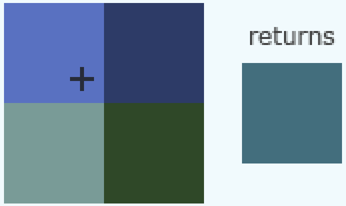
- Since one pixel does not usually corresponds directly to one texel, texture filtering is needed to determine the best colour for the pixel.
- OpenGL supports two texture filtering methods, namely:
  - **Nearest neighbour**
  - **Bilinear interpolation**

Texture Filtering | Nearest Neighbour



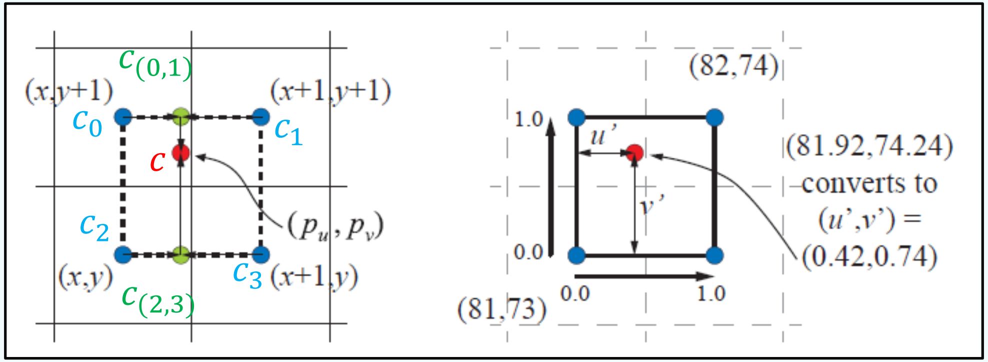
- Use the **nearest texel** that is closest to the texture coordinate.
- In the figure above, the upper-left texel has its centre closest to the texture coordinate.
- Therefore, it's chosen as a sampled colour.

Texture Filtering | Bilinear Interpolation



- Takes an interpolated value from the texture coordinate’s neighbouring texels, approximating the colour between the texels.
- The smaller the distance from the texture coordinate to a texel’s centre, the more that texel’s colour contributes to the sampled colour.

Texture Filtering | Bilinear Interpolation



$$u' = (p_u - 0.5) - \lfloor (p_u - 0.5) \rfloor$$
$$v' = (p_v - 0.5) - \lfloor (p_v - 0.5) \rfloor$$
$$c_{(0,1)} = (1 - u')c_0 + u'c_1$$
$$c_{(2,3)} = (1 - u')c_2 + u'c_3$$
$$c = (1 - v')c_{(2,3)} + v'c_{(0,1)}$$

Image taken from Figure 6.9 of Real-time Rendering (4th edition)

Texture Filtering | Bilinear Interpolation

GL_NEAREST

GL_LINEAR

Figure above shows the visual effect of each texture filtering method.

- **Nearest neighbor** – blocked patterns where texels that form the texture can be seen clearly
- **Bilinear Interpolation** – produces a smoother pattern where the individual pixels are less visible.

Q & A

Acknowledgement

- This presentation has been designed using resources from PoweredTemplate.com