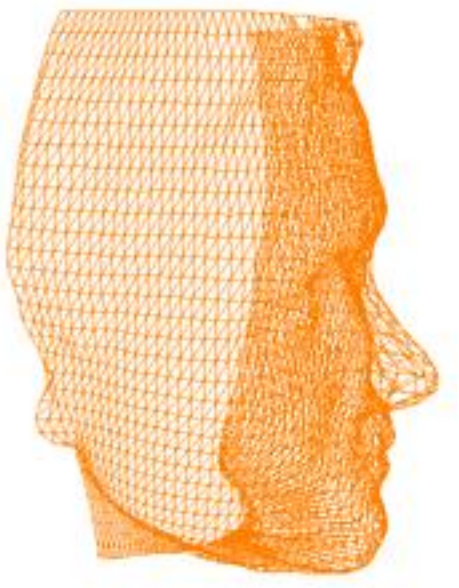
COSC422 Advanced Computer Graphics



6 Non-Photorealistic Rendering

Semester 2
2021

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Non-Photorealism – cartoonish

- A movement towards the generation of more creative and expressive imagery.
  - This rendering form is usually stylistic and incomplete, like traditional art forms.
  - Hand-drawn and hand-painted animations have an energetic quality that is lacking in computer rendered animation
- Advances in the area of shader programming (fragment and texture processing in fragment shaders) have sparked a growth in non-photorealistic rendering techniques and algorithms.
- Many traditional NPR methods[1] now have simple shader based implementations.

1. Gooch and Gooch, Non-Photorealistic Rendering, A.K. Peters, 2001.

Non-Photorealistic Rendering

Uses graphics algorithms to generate stylistic effects in the rendering of 3D models.

- **Toon-Rendering (Cel Shading)**: Cartoon style rendering of *3D objects* using special illumination models, highlighted edges and shadows.
- **Artistic Shading**: Hatching, Charcoal Rendering, Pencil shading etc., using textures and Tonal Art Maps.

Non-Photorealistic Rendering

Toon Shading
Two-Tone Shading

Cartoon

Graphics

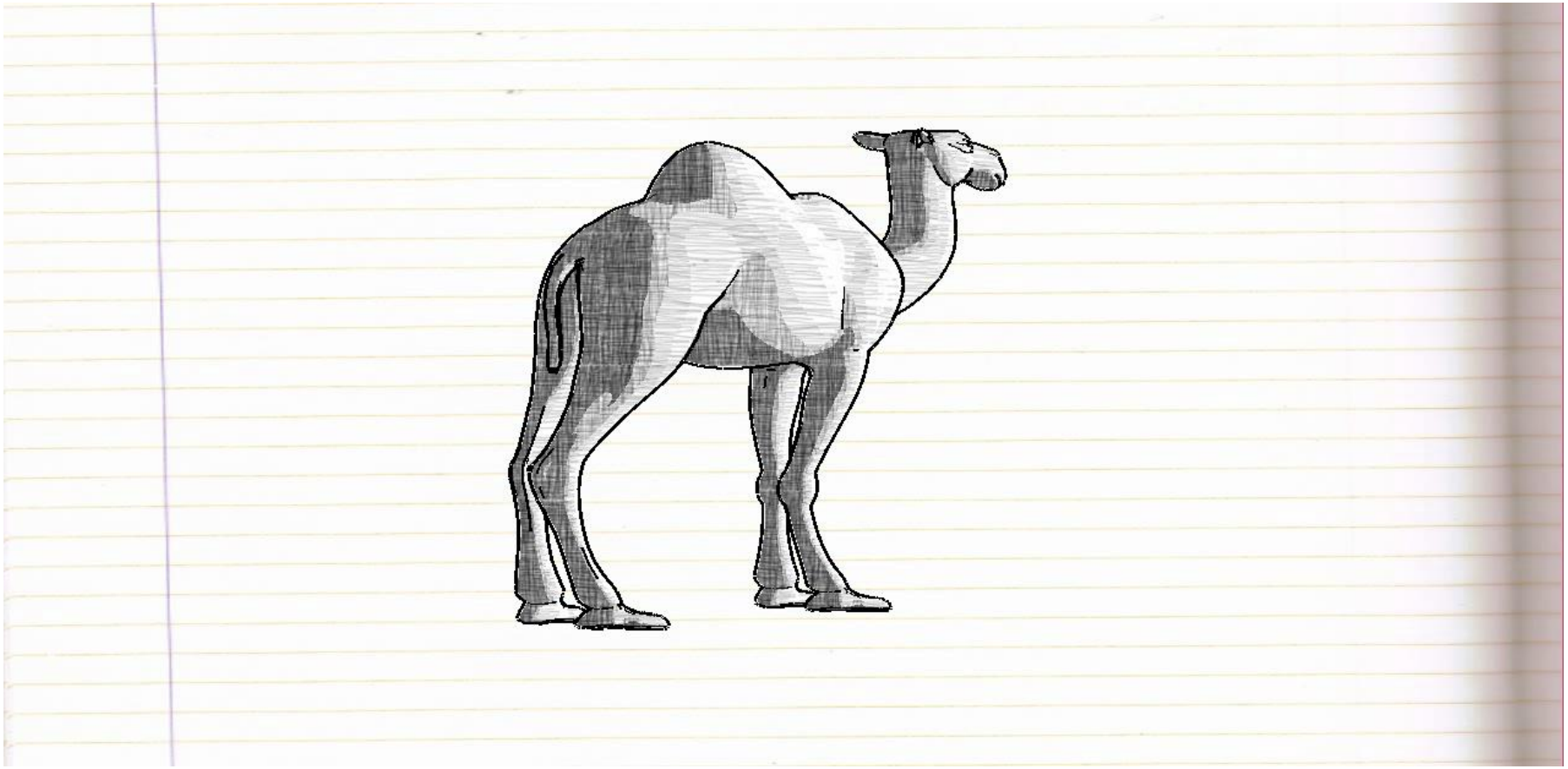
Ref:
Wikipedia: Marge Simpson
https://en.wikipedia.org/wiki/Marge_Simpson

Hatching/Pencil Shading

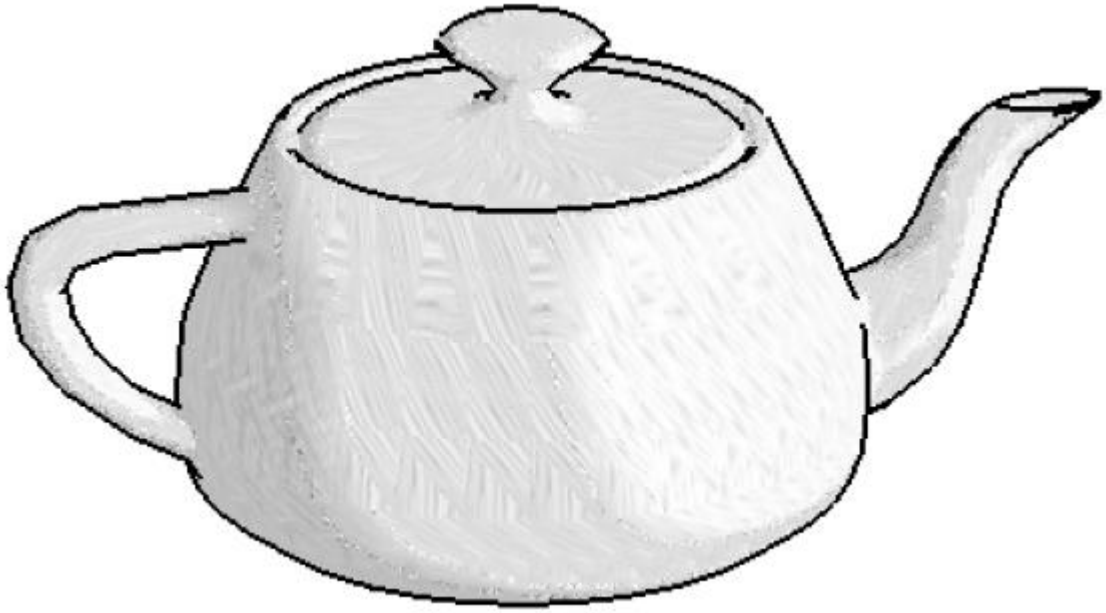
Pencil drawing
by an artist

3D Rendering using
Pencil-stroke textures

Non-Photorealistic Rendering

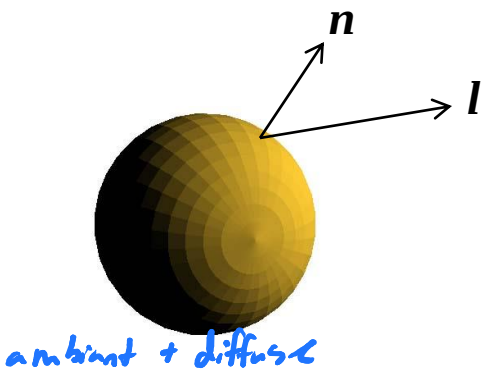


Non-Photorealistic Rendering



Ref: Programming Exercise 10

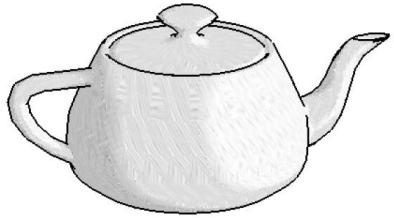
Shading Algorithms



- The most important parameter used in shading algorithms is the brightness factor (reflection from a diffuse material) given by $\boldsymbol{n.l}$

- **3-Tone Shading**: Use three tones of the same colour for specular, diffuse and ambient shades. color = ambient + diffuse + specular

- **Gooch Shading**: Modification of Phong model using two light sources to create a stylistic effect.

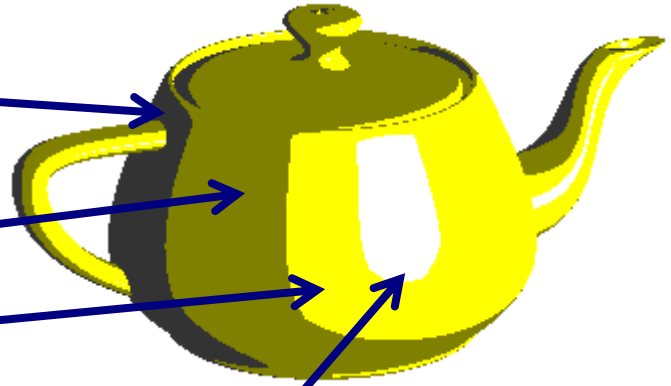


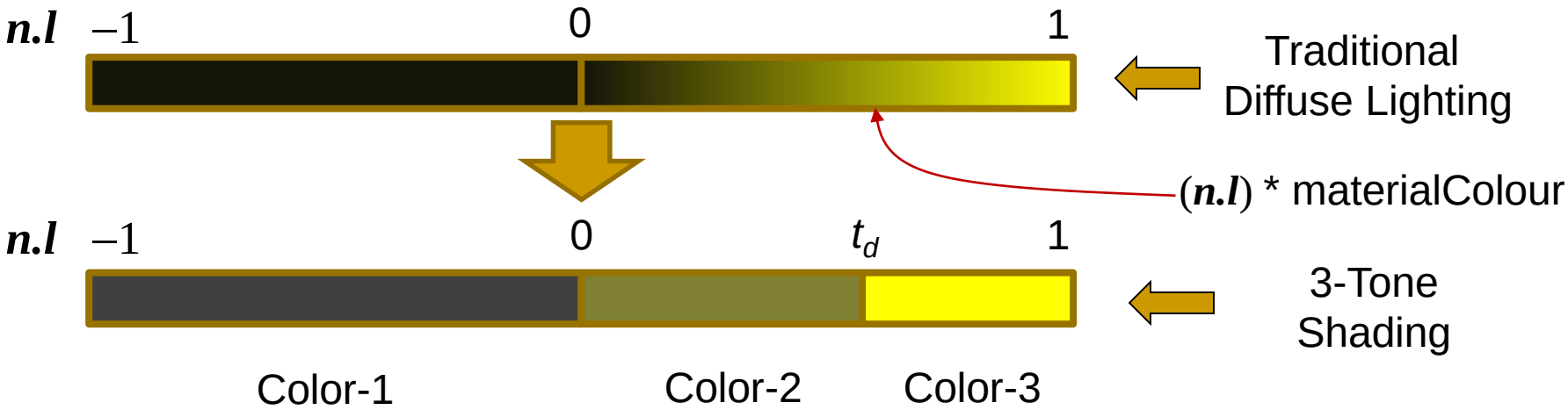
- **Pencil Shading**: Use $\boldsymbol{n.l}$ to select a 2D pencil-stroke texture with appropriate density.

3-Tone Shading

- Color-1 if $\boldsymbol{n}.\boldsymbol{l} \leq 0$ (shadow)
- Color-2 if $0 < \boldsymbol{n}.\boldsymbol{l} \leq t_d$ (mid-tone)
- Color-3 if $t_d < \boldsymbol{n}.\boldsymbol{l} \leq 1$ (highlight)
- (optional)

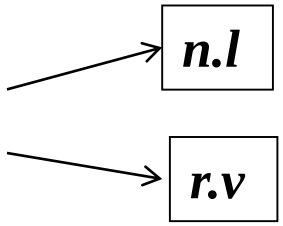
  White if $t_s < \boldsymbol{r}.\boldsymbol{v} \leq 1$ (specular highlight)





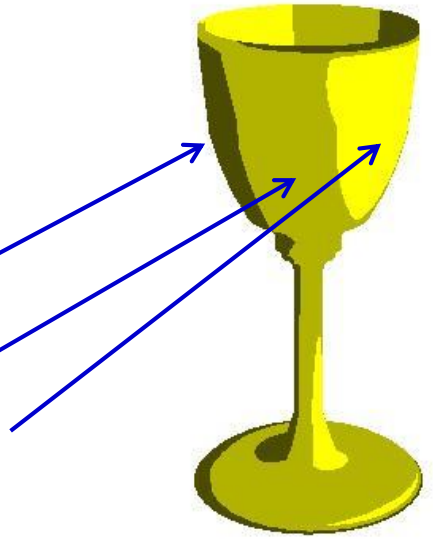
3-Tone Shading

Using a programmable pipeline (OpenGL-4), we can easily implement 3-tone shading in the **fragment shader**.




```
in float diffTerm;
in float specTerm;
out vec4 outColor;

void main()
{
    if(specTerm > 0.98) outColor = vec4(1.0);
    else if(diffTerm < 0.0)
        outColor = vec4(0.2, 0.2, 0.2, 1.0);
    else if(diffTerm < 0.6)
        outColor = vec4(0.5, 0.5, 0.0, 1.0);
    else outColor = vec4(1.0, 1.0, 0.0, 1.0);
}
```

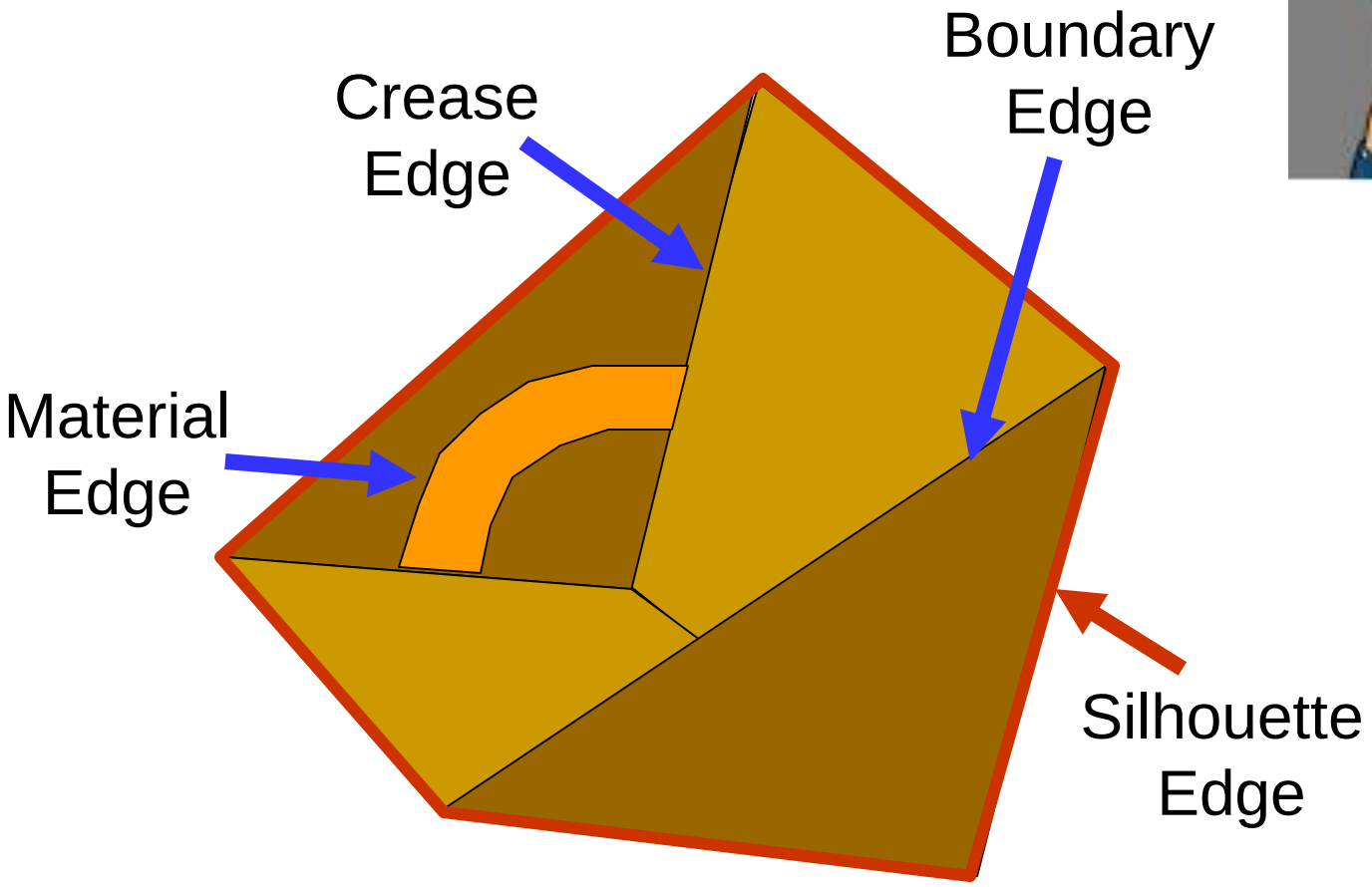


Importance of Edges

Edges play an important role in toon-shading algorithms.

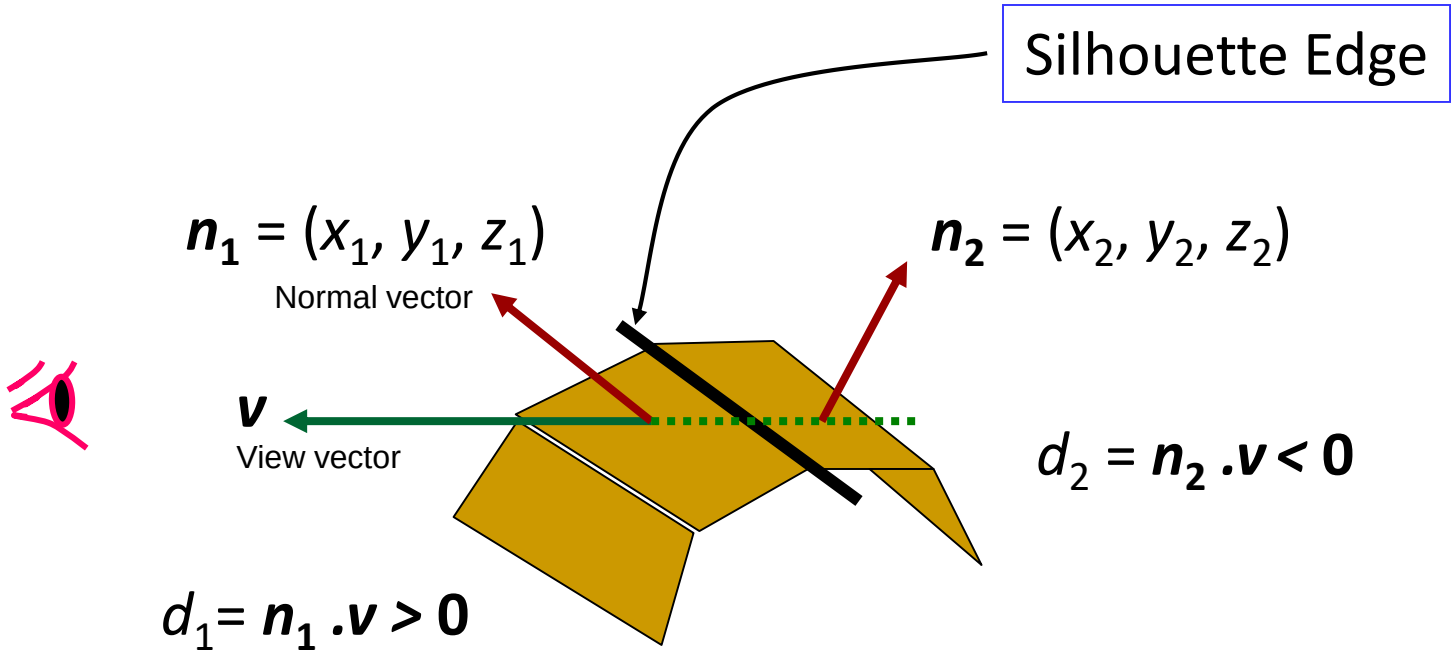
- **Silhouette edges:** Separates an object from the background.
- **Boundary/Border edges**: Edges that are not shared by two polygons.
- **Crease edges**: Edges with sharp discontinuities in the normal directions.
- **Material edges**: Edges between polygons with different material properties.

Types of Edges



Silhouette Edges

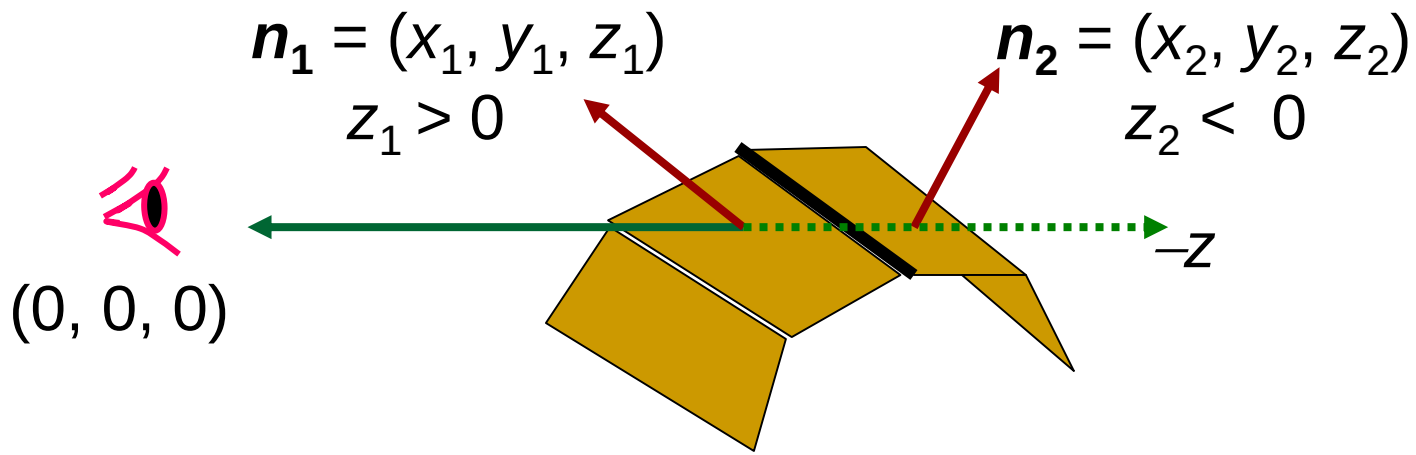
A silhouette edge is an edge between a front-facing and a back-facing polygon.



World Coordinate Space

Silhouette Edges

If the surface normal vectors have already been transformed into eye coordinate space, then the silhouette edge can be detected using only the *z*-components of the normal vectors.



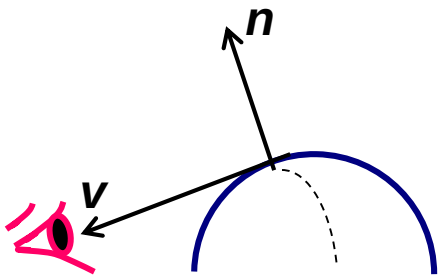
Eye Coordinate Space

Silhouette Edges

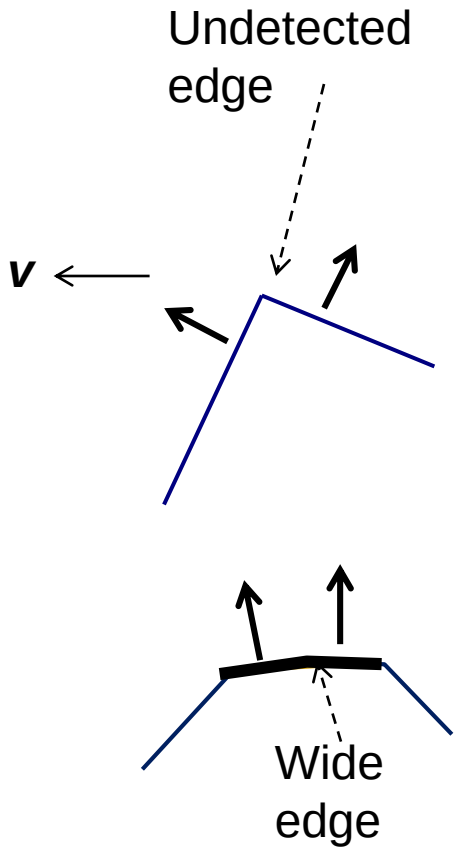
- The previous method requires information about adjacent triangles bordering each edge of a primitive.
  - We can effectively process such information using **geometry shaders** (discussed later)

- On a smooth surface, a silhouette edge corresponds to *fragments* where $\boldsymbol{n}.\boldsymbol{v} = 0$.
  - Easily implemented in a fragment shader
  - But, not very effective!

    (see next slide)



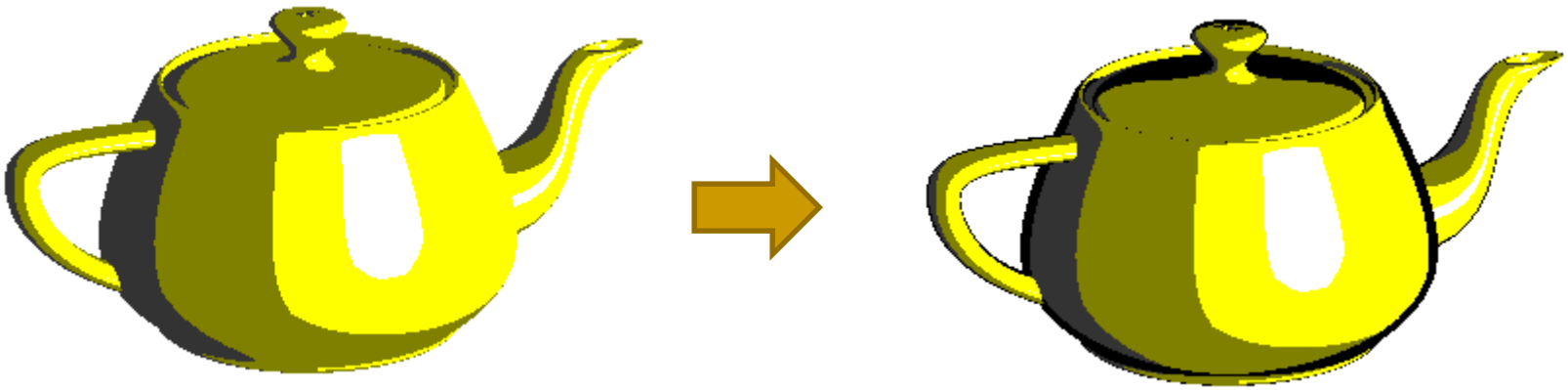
Silhouette Edges Using n.v



- Fragment shader implementation:

```
If (abs(dot(normal, view)) < epsilon)
                    outColor = vec4(0);
```

- Problems:
  - Edge thickness varies with curvature
  - Some edge pixels may not be visible!

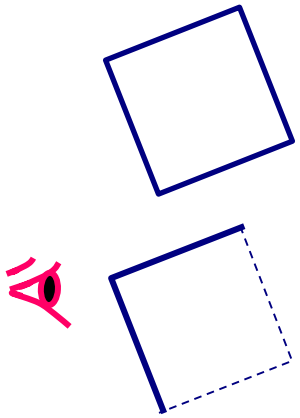


Silhouette Edges (Two-pass Rendering)

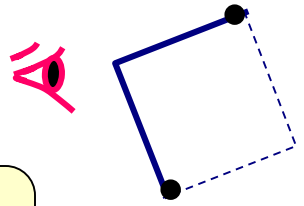
```
glEnable(GL_CULL_FACE);

glPolygonMode(GL_FRONT, GL_FILL);
glDepthFunc(GL_LESS);
glCullFace(GL_BACK);
glDrawElements(...);

glPolygonMode(GL_BACK, GL_LINE);
glDepthFunc(GL_LEQUAL);
glCullFace(GL_FRONT);
glDrawElements(...);
```



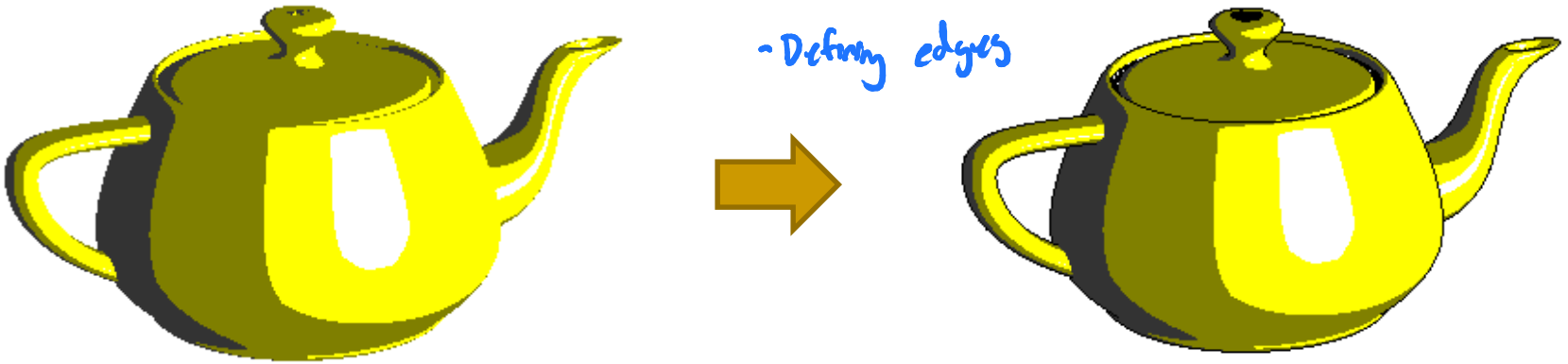
Draw and fill front-facing polygons



Draw back-facing polygons in line mode.

Set colour to black !

Silhouette Edges (Two-pass Rendering)



Advantages:

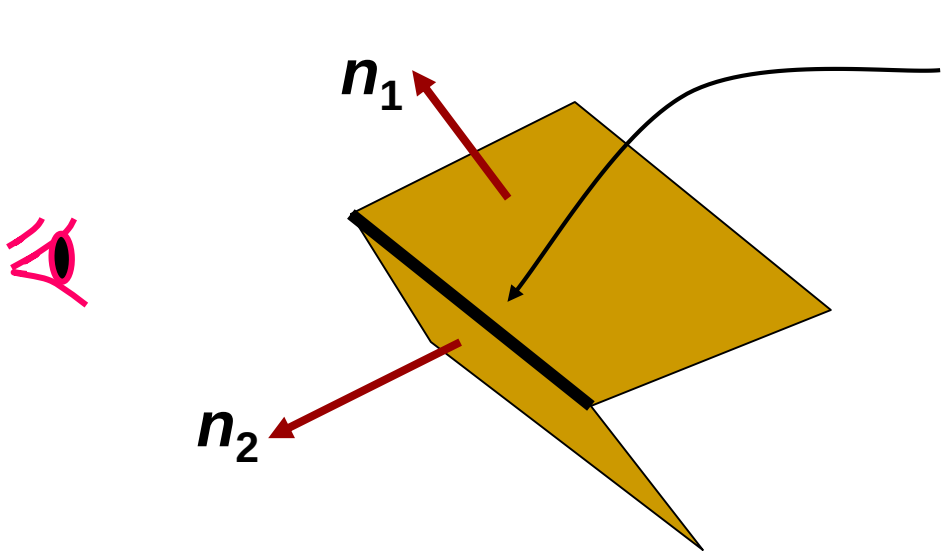
- Very efficient algorithm for silhouette edge rendering
- Can be implemented easily in the fixed function pipeline.
- Does not require any additional geometry processing

Drawback:

- The model needs to be rendered twice

Crease / Hard Edge

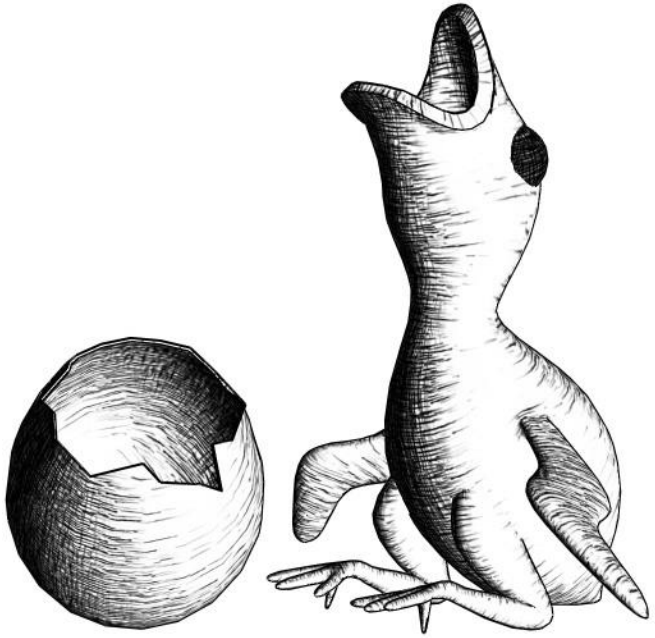
A crease is an edge shared by two polygons, where the angle between the two normal vectors is greater than a predefined threshold θ_{thresh}



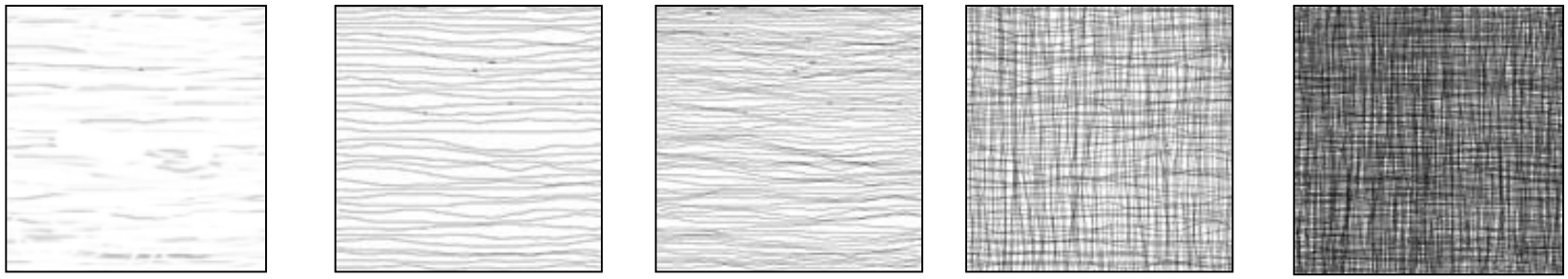
θ: The dihedral angle.
For a crease,
$\theta > \theta_{thresh}$
$\boldsymbol{n_1 . n_2 < \cos(\theta_{thresh}) = T}$

Pencil Shading

Use ***n.l*** to select an appropriate pencil stroke texture.



Pencil stroke textures with increasing density values:

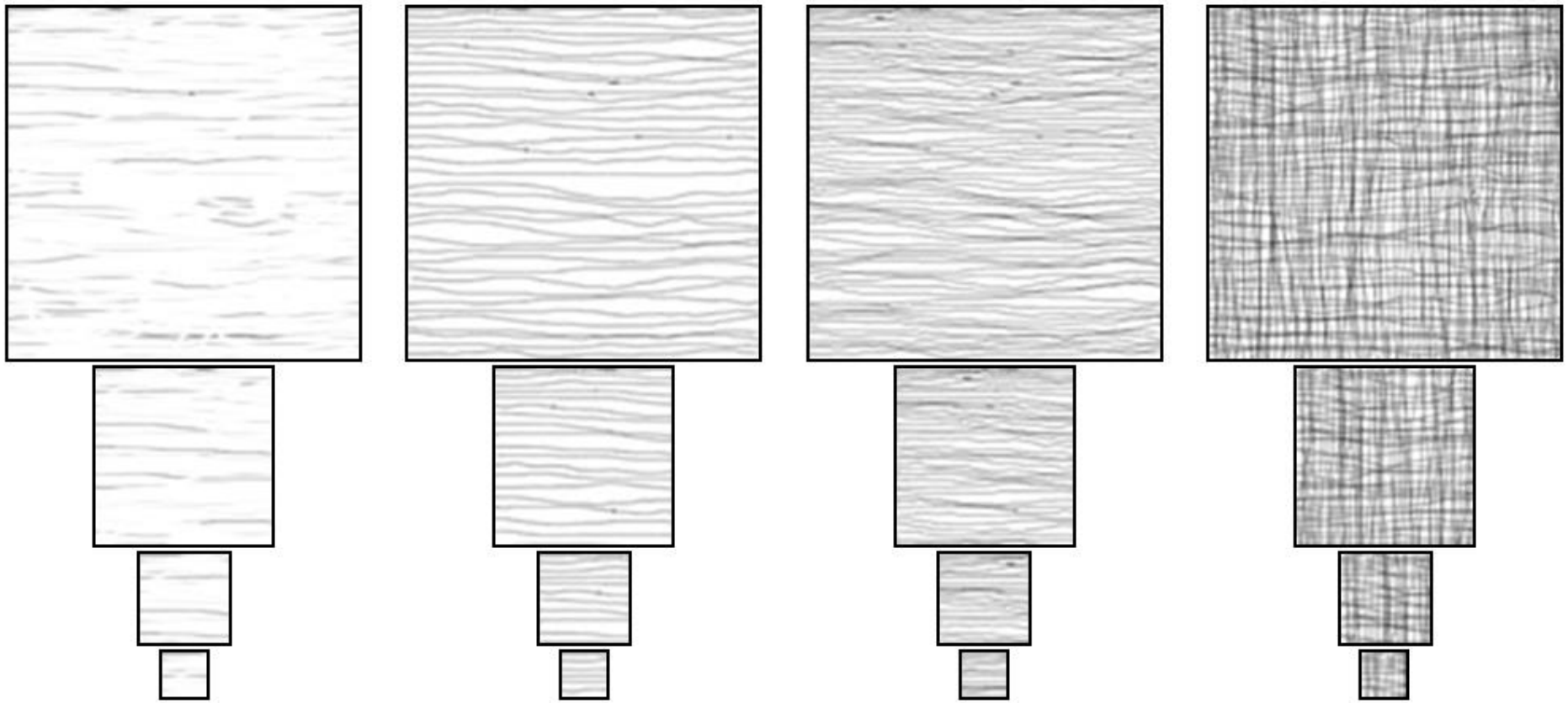


n.l ∈[0.8, 1.0] ***n.l*** ∈[0.5, 0.8] ***n.l*** ∈[0.3, 0.5] ***n.l*** ∈[0, 0.3] ***n.l*** ∈[–1, 0]

Ref:
Gooch and Gooch, Non-Photorealistic Rendering, A.K. Peters, 2001.

Tonal Art Map (TAM)

Mip-map sets of textures with varying stroke densities

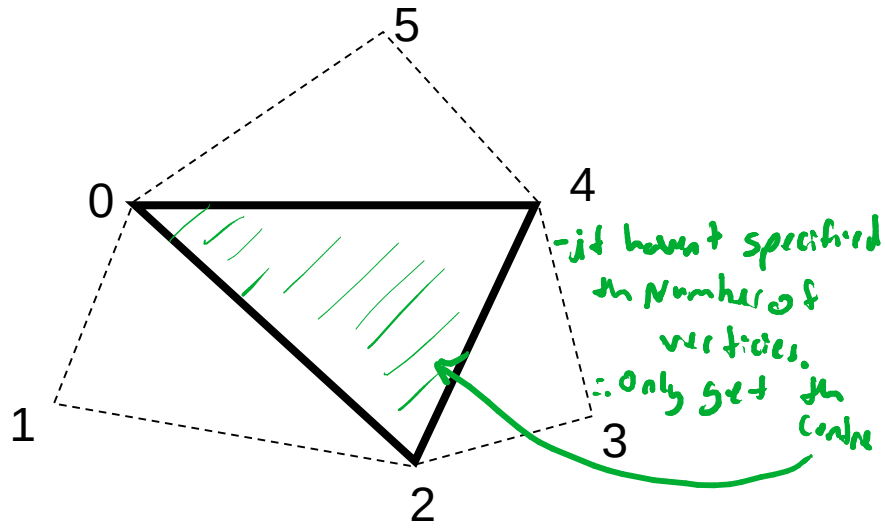


A mip-map set is required to maintain the stroke density of the selected texture independent of the projected area of polygons.

The Adjacency Primitive

The primitive type GL_TRIANGLES_ADJACENCY is very useful for non-photorealistic rendering applications.

- 6 elements for each triangle
- The adjacency information can be used to detect silhouette and crease edges on each triangle.



Geometry shader:

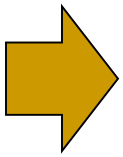
```
gl_in[i].gl_Position
```

0...5

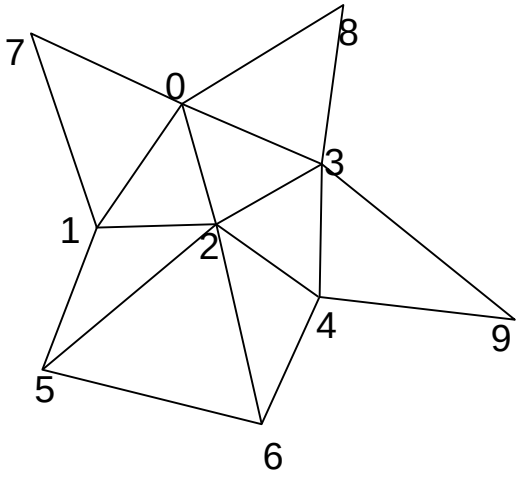
The Adjacency Primitive

- The element array for the adjacency primitive must be defined by the user.

```
0  1  2
0  2  3
3  2  4
2  6  4
5  6  2
2  1  5
1  0  7
0  3  8
3  4  9
```



```
0  7  1  5  2  3
0  1  2  4  3  8
3  0  2  6  4  9
```

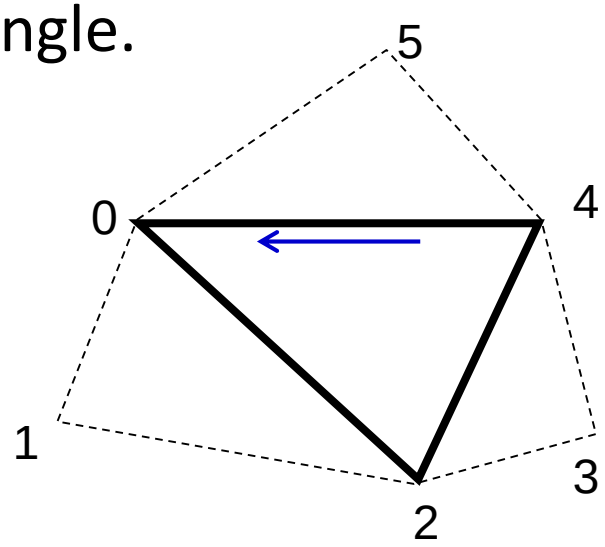



```
glDrawElements
      ( GL_TRIANGLES,
        npoly * 3,
        GL_UNSIGNED_SHORT,
        NULL );
```

```
glDrawElements
      ( GL_TRIANGLES_ADJACENCY,
        npoly * 6,
        GL_UNSIGNED_SHORT,
        NULL );
```

The Adjacency Primitive

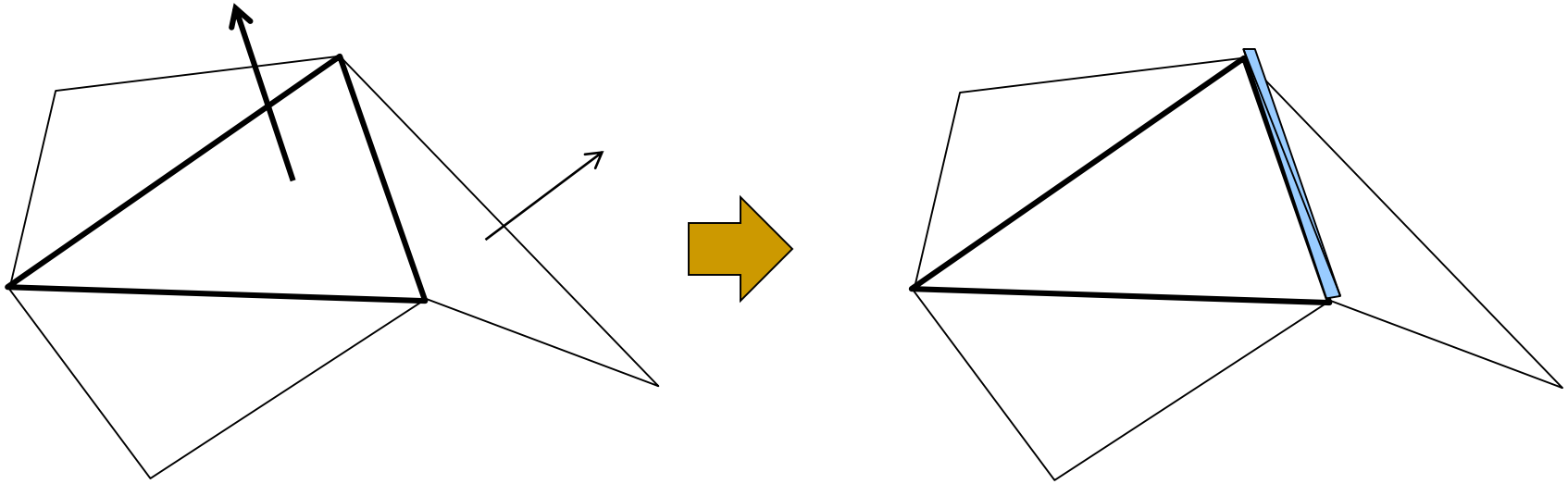
- We require a suitable mesh data structure such as the **half-edge structure** for obtaining the adjacency information for each triangle.



- The process of creating the element array for adjacency primitives will be discussed in the next section on mesh processing algorithms.

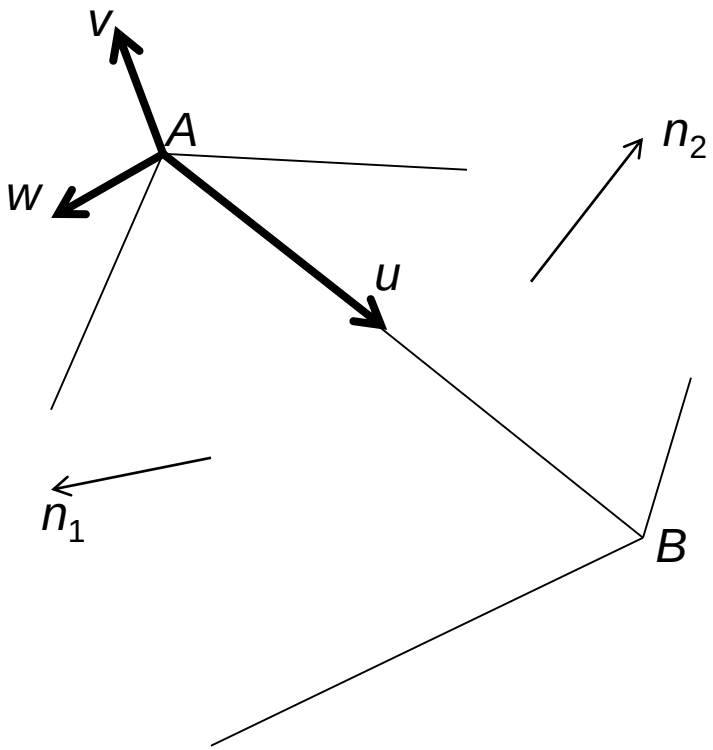
Crease Edge Identification

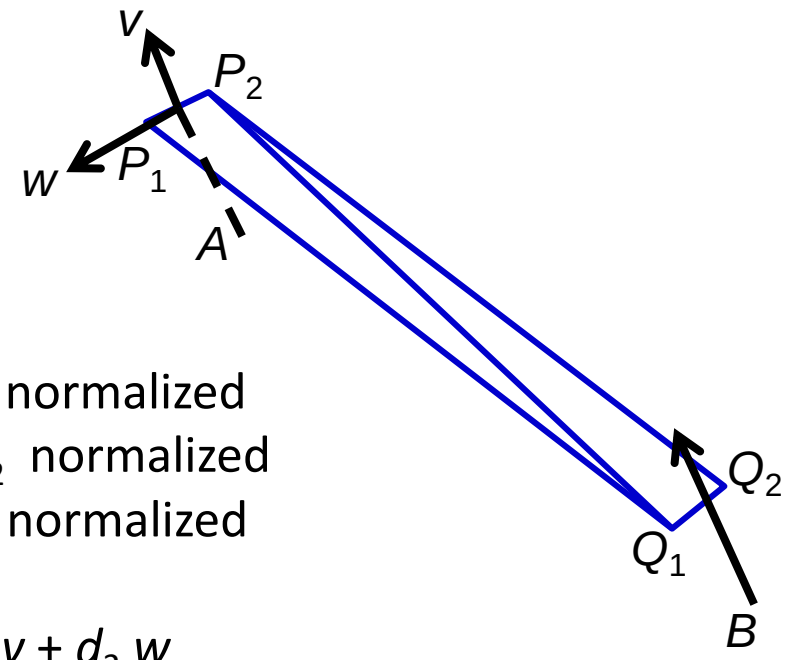
- Inside the geometry shader, we can compute the face normals of the current triangle and its three adjacent triangles.
- If the angle between two adjacent face normals is greater than a threshold, the edge between the two faces can be classified as a crease edge.



Crease Edge Rendering

A crease edge is rendered as a thin triangle strip consisting of two triangles, positioned slightly above the edge.





$u = B - A$ normalized
$v = n_1 + n_2$ normalized
$w = u \times v$ normalized

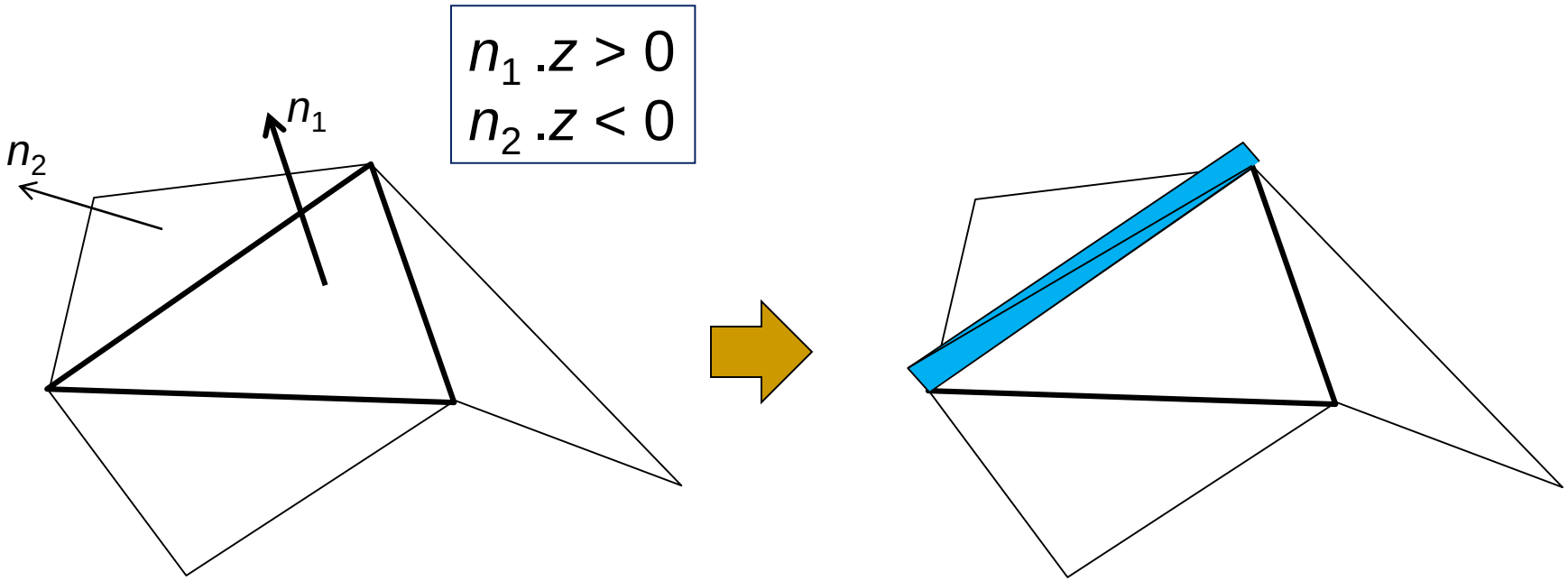
$P_1 = A + d_1 v + d_2 w$
$P_2 = A + d_1 v - d_2 w$ etc.

d_1, d_2 are small offset values

Silhouette Edge Identification

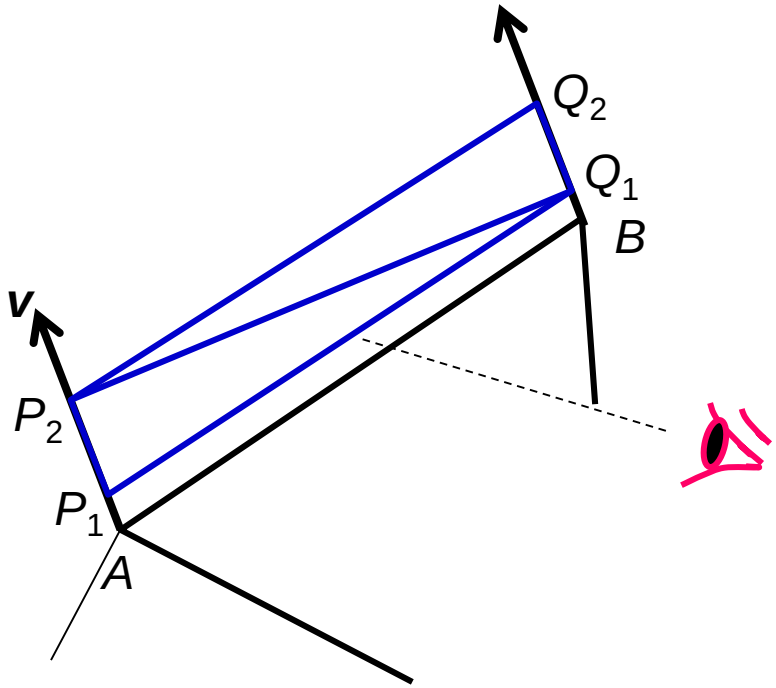
A silhouette edge is an edge between a front-facing triangle and a back-facing adjacent triangle.

The surface normal vectors are assumed to be in the eye-coordinate space.



Silhouette Edge Rendering

Just like crease edges, silhouette edges are also rendered as thin triangle strips, but oriented towards the viewer.



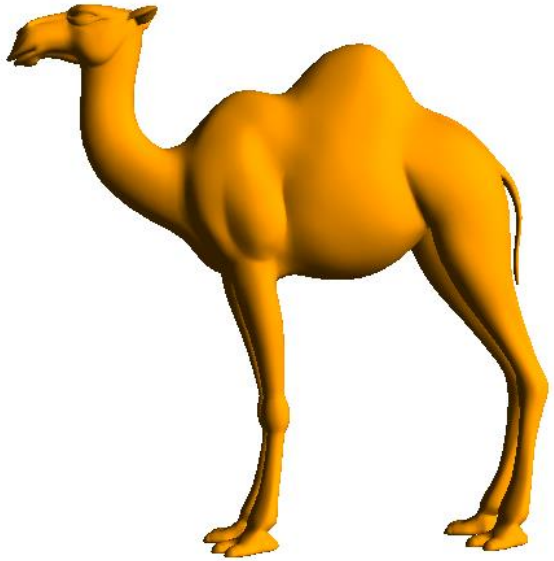
$v = n_1 + n_2$ normalized

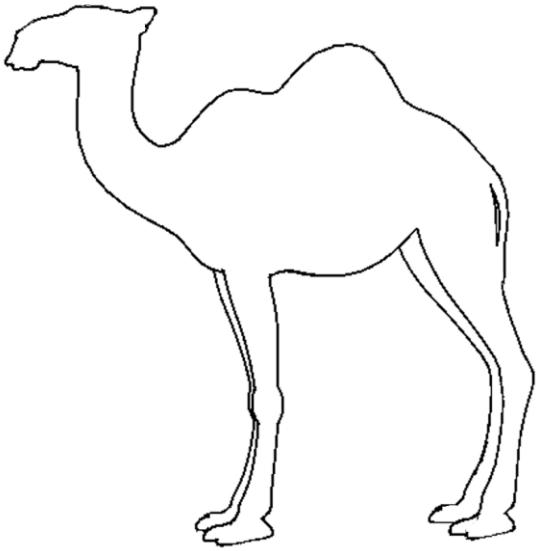
$P_1 = A + d_1 v$

$P_2 = A + d_2 v$ etc.

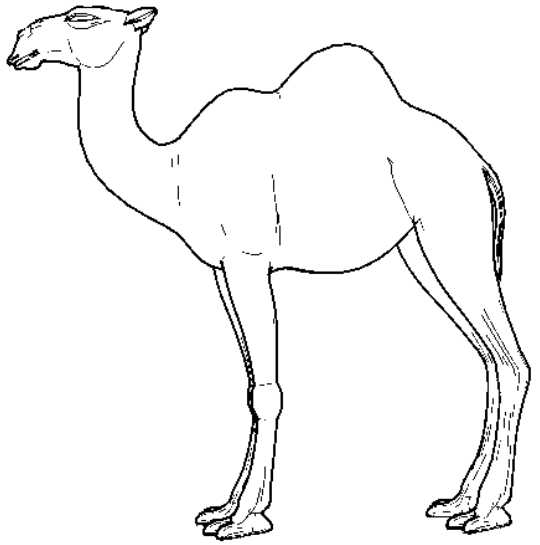
d_1, d_2 are small offset values

Edge Rendering in Geometry Shader





Silhouette Edges

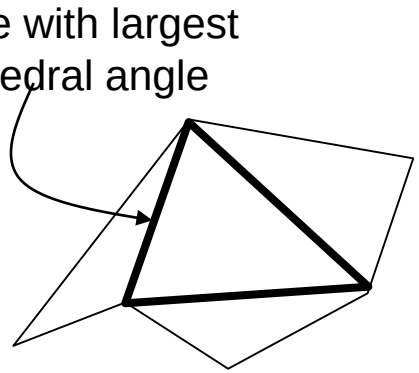


Silhouette + Crease Edges

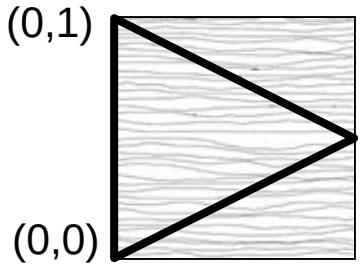
Pencil Shading

- Use four or five pencil shade textures with varying stroke density. Generate their mipmap sets to form a tonal art map (TAM). Select appropriate texture for mapping inside the fragment shader, based on the value of ***n.l***
- Assign texture coordinates to each triangle inside the geometry shader based on local curvature

Texture



Triangle Adjacency
Primitive



Our prior work on NPR

- J. Shin, M. Haller, R. Mukundan, “A Stylized Cartoon Hair Renderer”, Proc. of ACM SIGCHI International Conference on Advances in Computer Entertainment Technology - ACE 2006, 14-16 June, Hollywood, USA. ACM Press.

- R. Mukundan, “Multilevel Stroke Textures for Sketch Based Non-Photorealistic Rendering”, Proc. of 6th IEEE Intnl. Conf. and Workshop on Computing and Communication – IEMCON15, University of British Columbia, 15-17 Oct 2015.

- Putri, T., Mukundan, R., Neshtatian, K, "Artistic style characterization and brush stroke modelling for non-photorealistic rendering", International Conference on Image and Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4-6 Dec 2017.

- Putri, T. and Mukundan, R., “Iterative Brush Path Extraction Algorithm for Aiding Flock Brush Simulation of Stroke-Based Painterly Rendering”. In Evolutionary and Biologically Inspired Music, Sound, Art and Design. Springer International Publishing, Mar 2016, pp. 152-162.