CPSC 591 Project

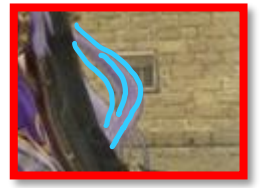
Feature Lines + Attribute Based Mapping

Motivation & Goal

Motivation & Goal

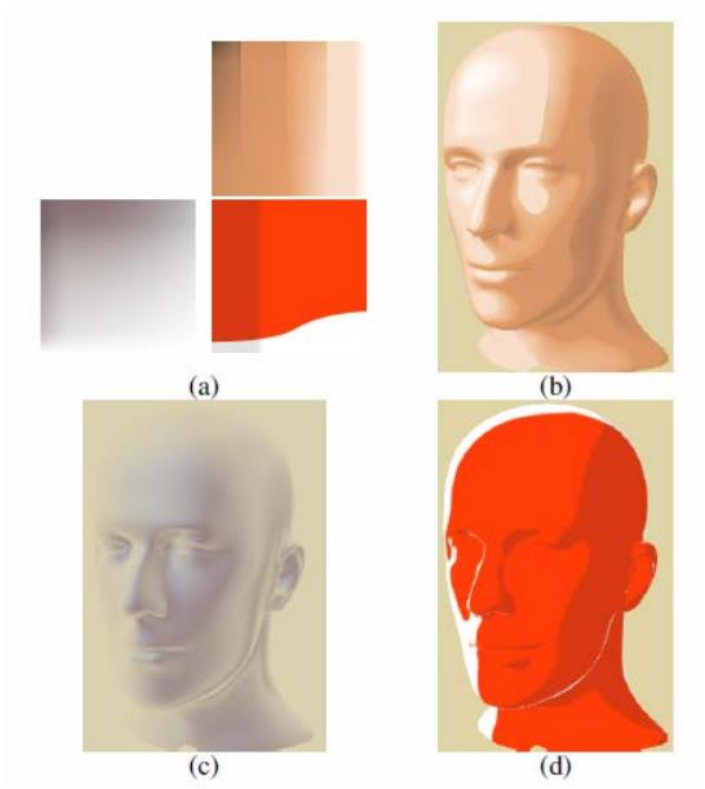
Silhouette & crease

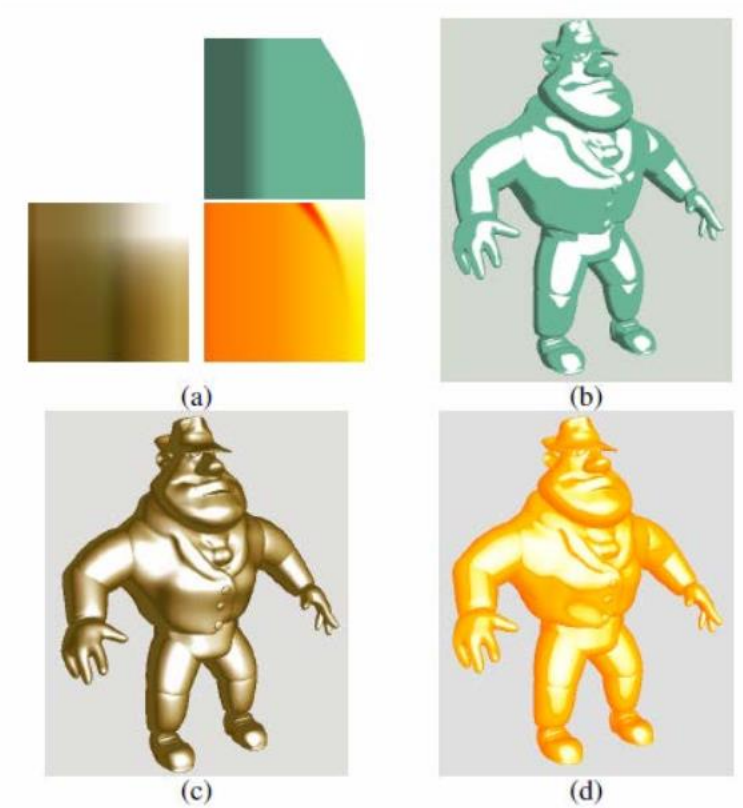
Silhouette & crease



Motivation & Goal

From lecture, we learned attribute-based mapping.

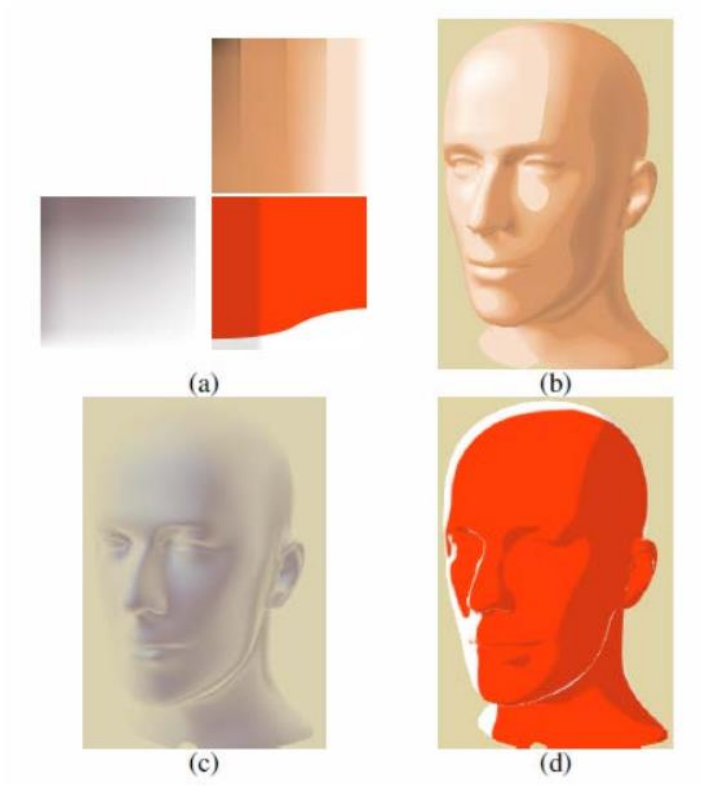


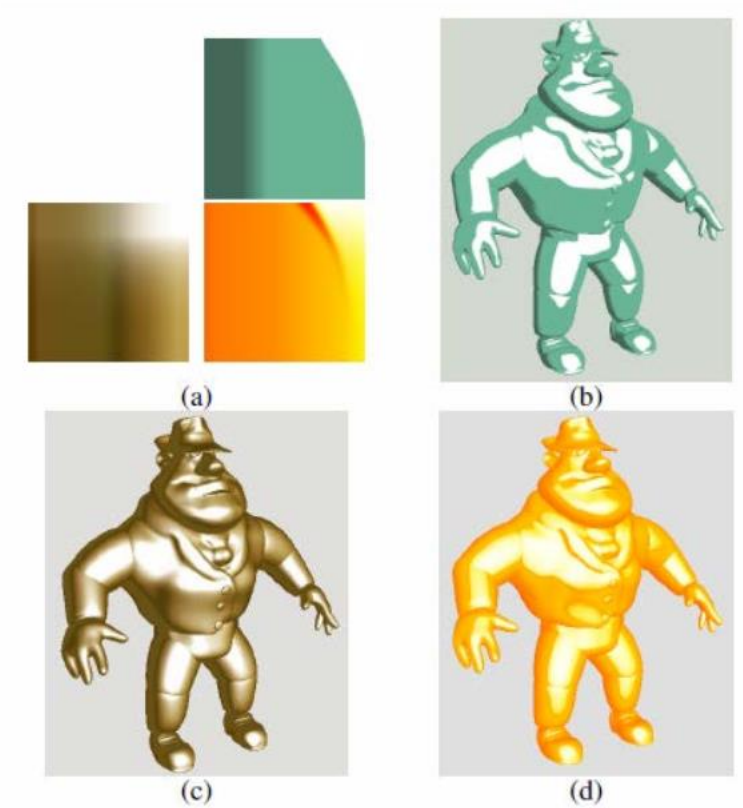


However, we need to generate different textures for different color.

Motivation & Goal

From lecture, we learned attribute-based mapping.





Goal: user can select input color

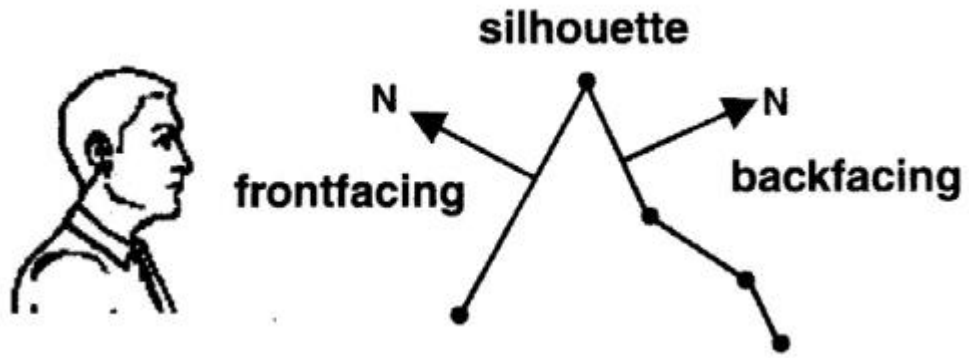
However, we need to generate different textures for different color.

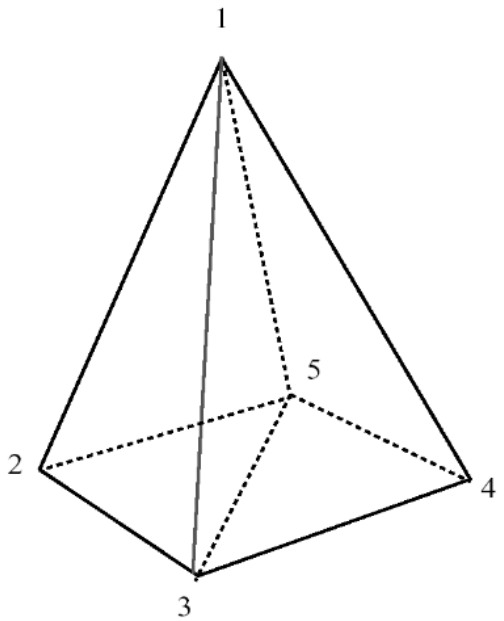
Feature Lines

1. Silhouette
2. Crease

Silhouette

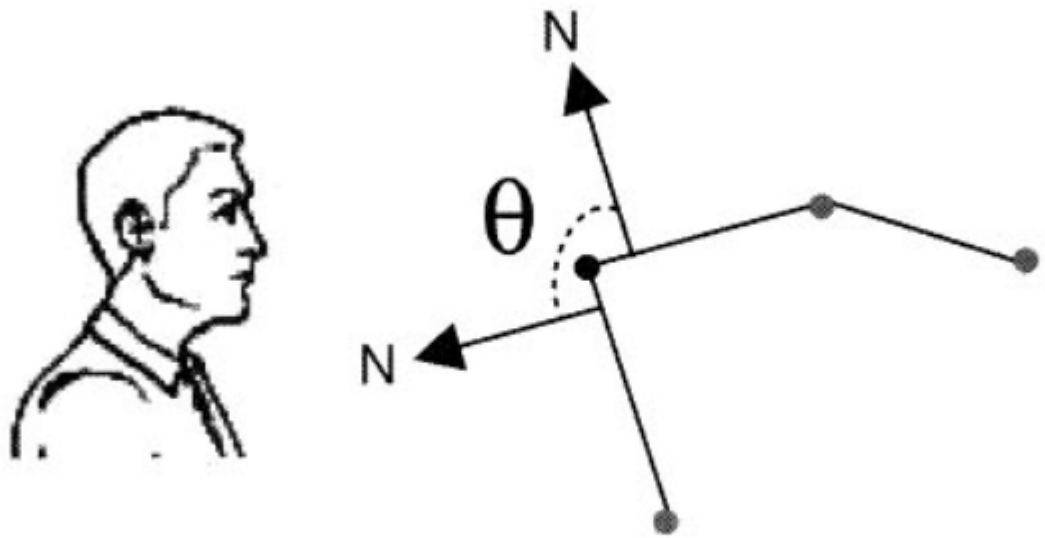
Algorithm from the lecture, using edge buffer

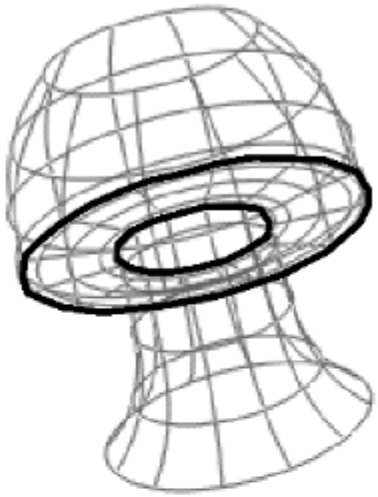




Vertex	*VFB*	*VFB*	*VFB*	*VFB*
1	211	300	411	500
2	311	500	x00	x00
3	411	500	x00	x00
4	500	x00	x00	x00
5	x00	x00	x00	x00

Interior : Model/Threshold → Crease

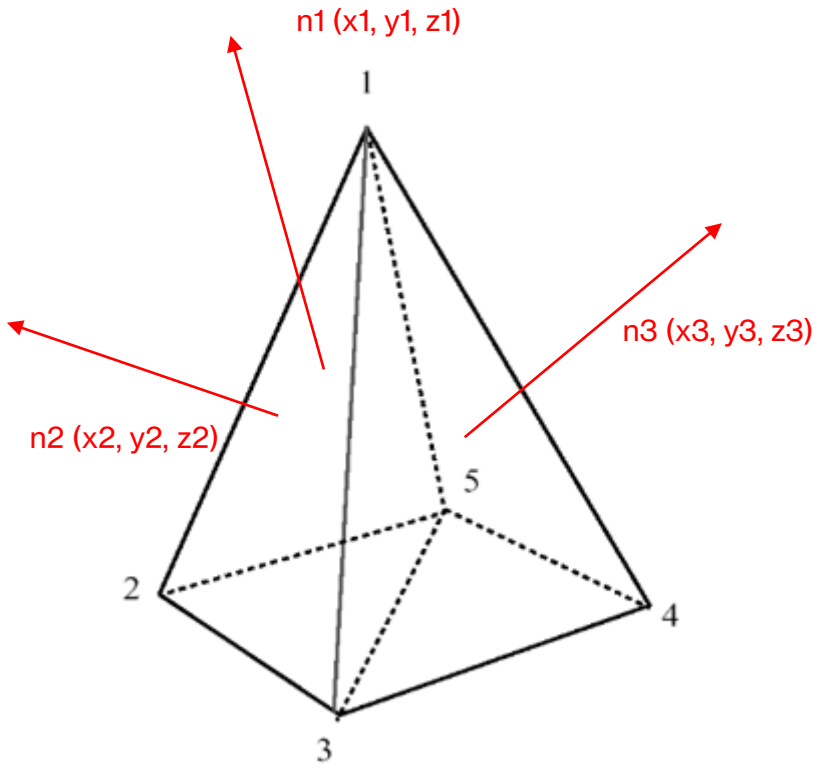




Polygonal model: an edge between two front-facing polygons whose dihedral angle theta is above some threshold

Also using edge buffer

Crease

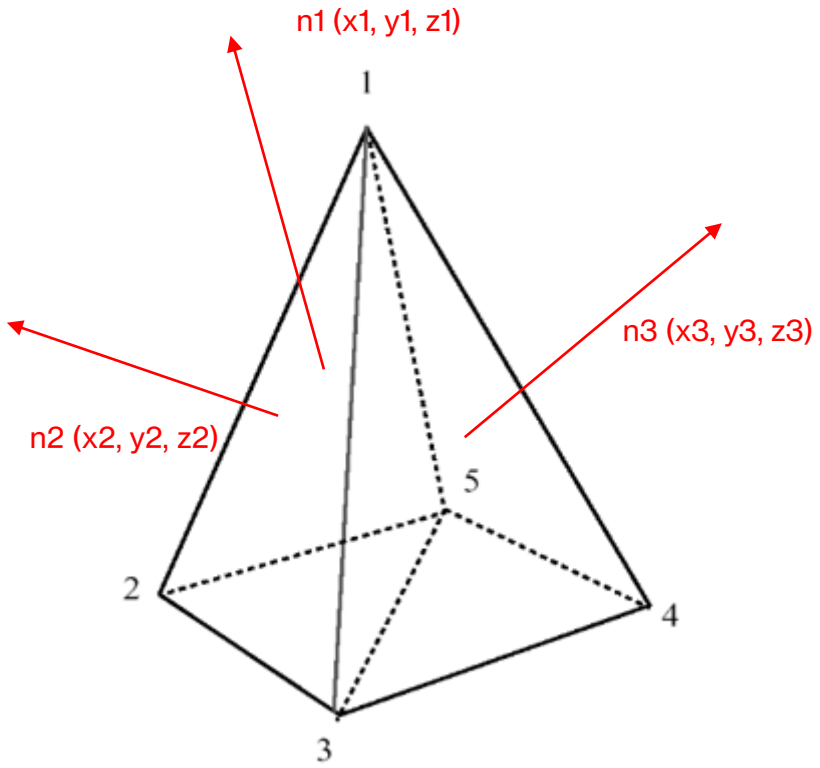


New edge buffer

Vertex	VFB Ns	VFB Ns	VFB Ns	VFB Ns
1	211 n1 n2	300 n1 n3	...	...
2	...	...	...	...
3	...	...	...	...
4	...	...	...	...
5	...	...	...	...

1 edge is shared by 2 faces,
so it has 2 normal for each face

Crease



1 edge is shared by 2 faces,
so it has 2 normals

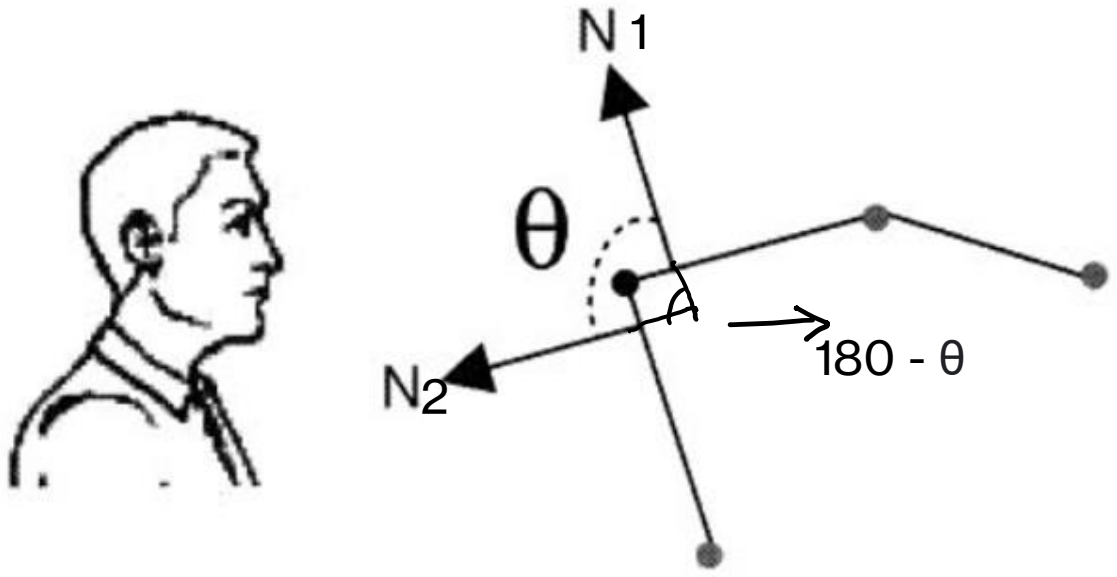
New edge buffer

Vertex	VFB Ns	VFB Ns	VFB Ns	VFB Ns
1	211 n1 n2	300 n1 n3	...	...
2	...	...	...	...
3	...	...	...	...
4	...	...	...	...
5	...	...	...	...

Data structure

```
//adjacent list
struct Node {
    unsigned int v;                 //vertex id(index)
    unsigned int f = 0;             //front face bit
    unsigned int b = 0;             //back face bit
    std::vector<glm::vec3> norms;   //normals of 2 faces
};
```

Crease

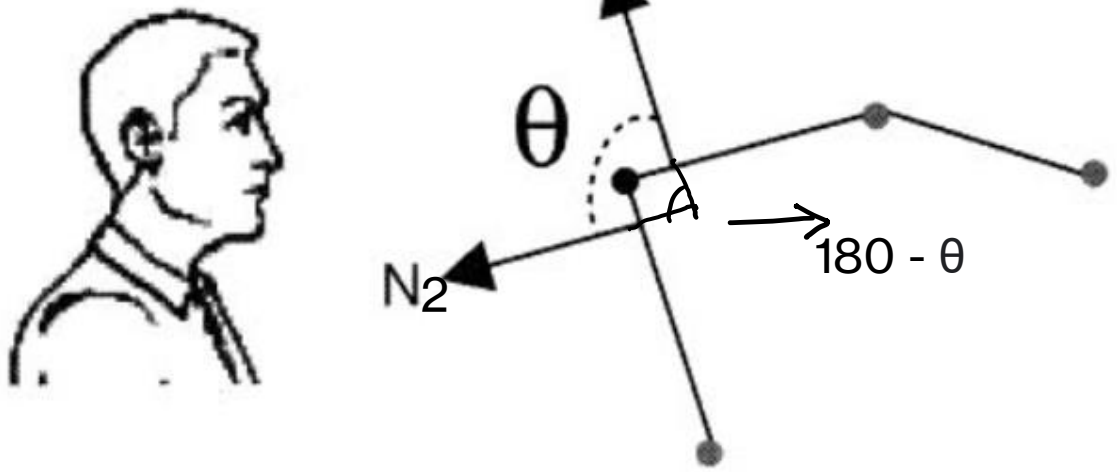


If theta >= threshold
render crease

180 – theta <= 180 – threshold

Dot(N1, N2) <= cos(180 - threshold)

Crease



Only need to check front faces
If theta >= threshold
render crease

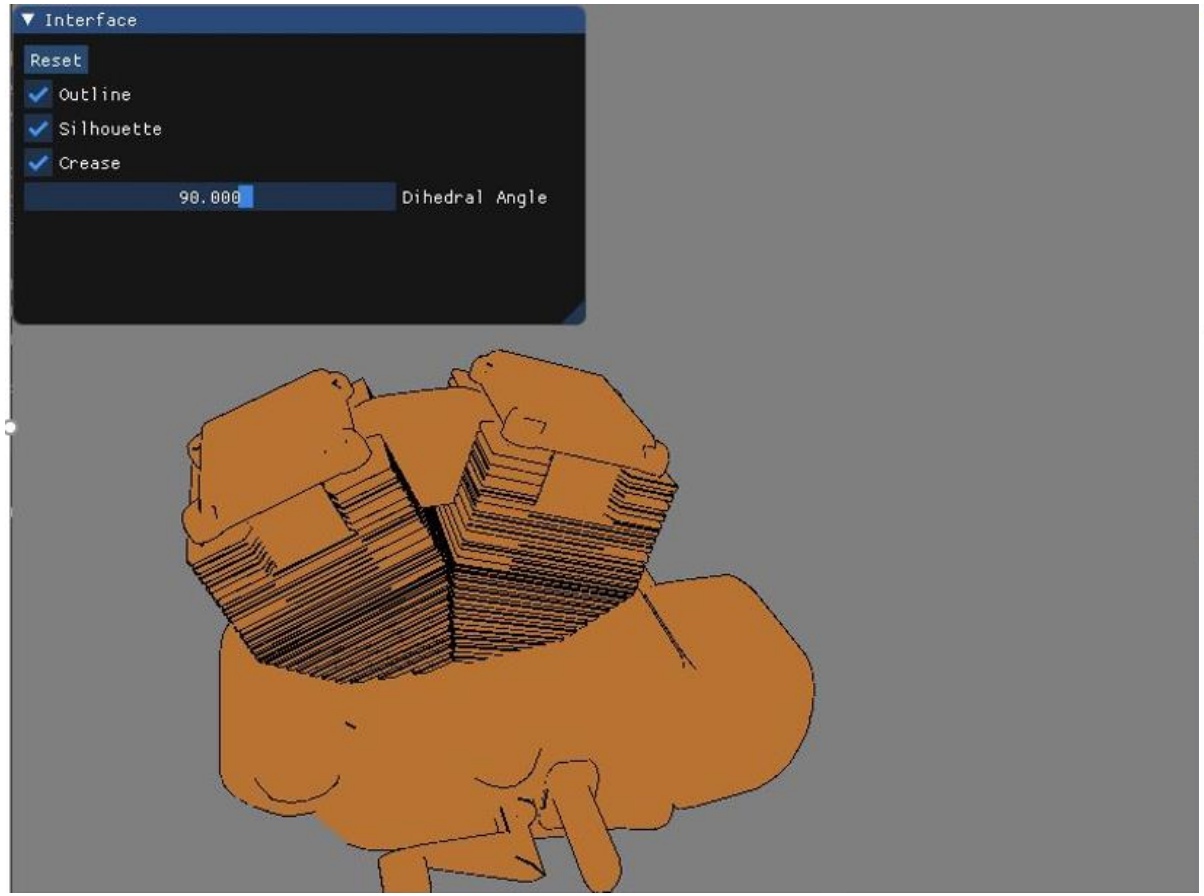
180 – theta <= 180 – threshold

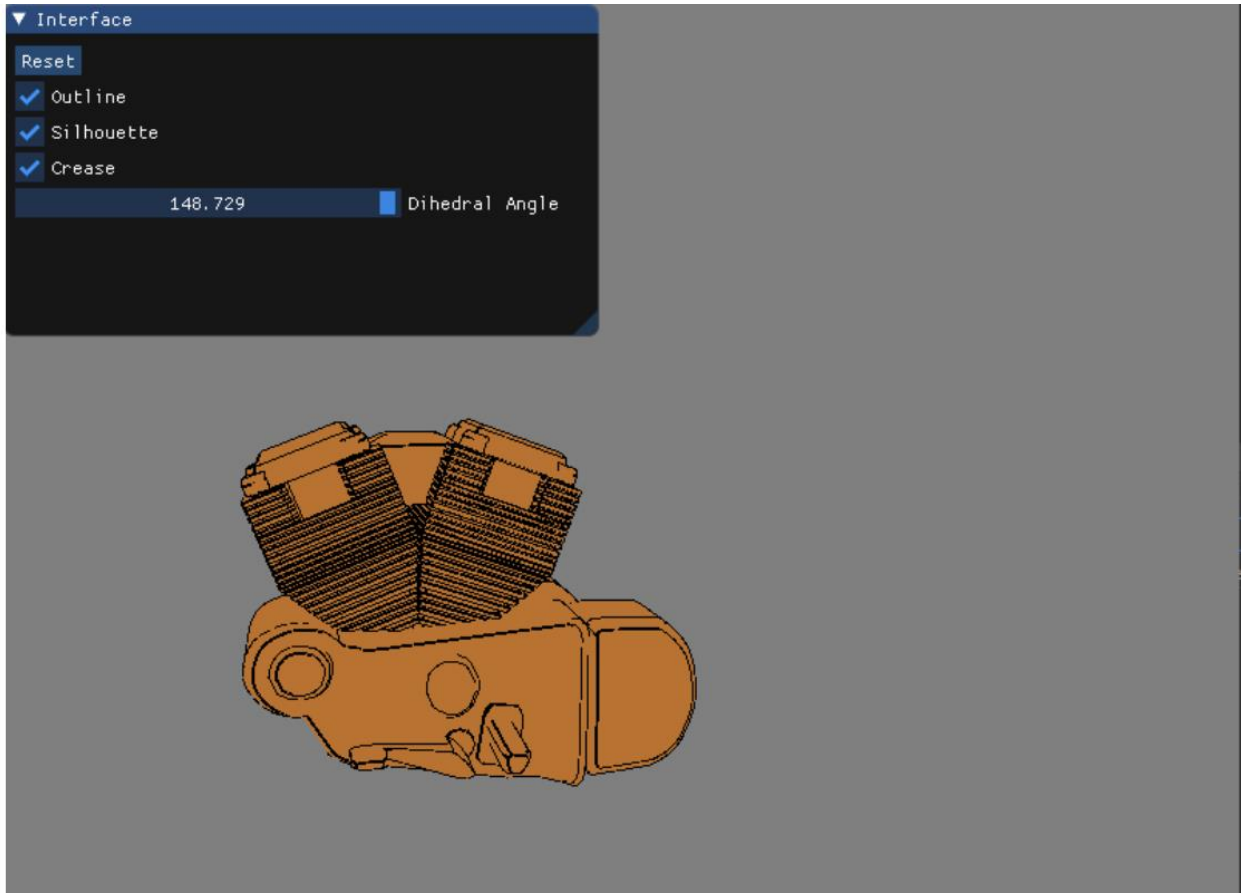
Dot(N1, N2) <= cos(180 - threshold)

code

```
for (int j = 0; j < edgeBuffer.size(); ++j) {
    Vertex v0, v1;
    for (auto it = edgeBuffer[j].begin(); it != edgeBuffer[j].end(); ++it) {
        //if it's front face
        if (!(*it).f && !(*it).b) {
            v0 = ourModel.meshes[0].vertices[j];
            v1 = ourModel.meshes[0].vertices[(*it).v];
            if (it->norms.size() >= 2) {
                glm::vec3 norm1 = it->norms[0];
                glm::vec3 norm2 = it->norms[1];
                if (glm::dot(norm1, norm2) <= glm::cos(glm::radians(180.f - creaseAngle))) {
                    vertices.push_back(v0.Position);
                    vertices.push_back(v1.Position);
                }
            }
        }
    }
```

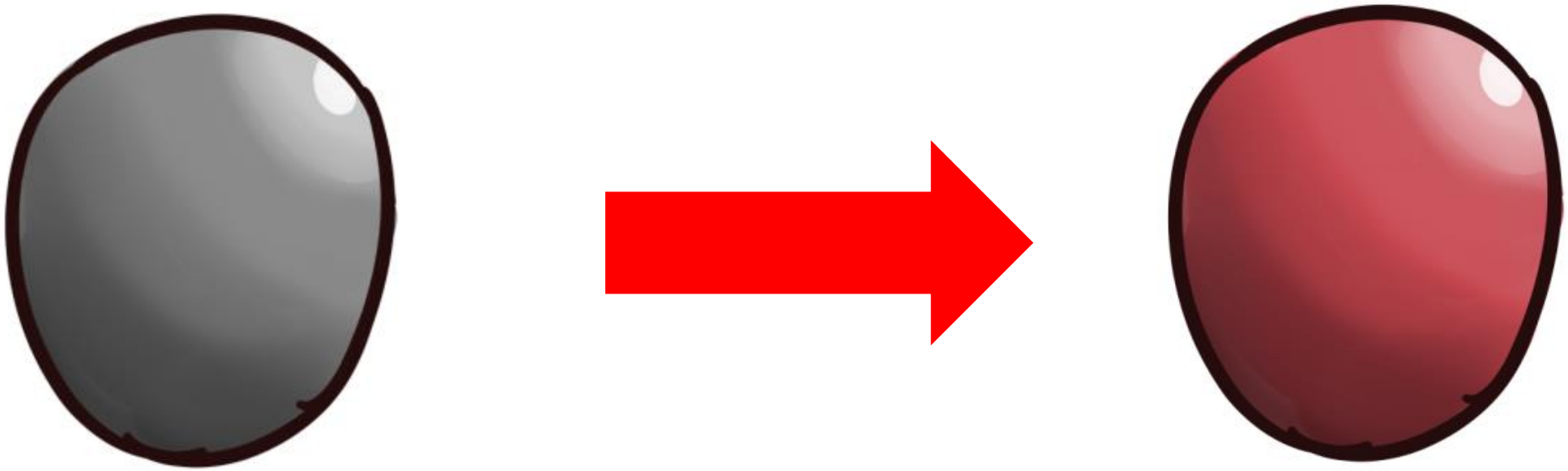
Result


Interface
Reset
Outline
Silhouette
Crease
90.000
Dihedral Angle


Interface
Reset
Outline
Silhouette
Crease
148.729
Dihedral Angle

Texture

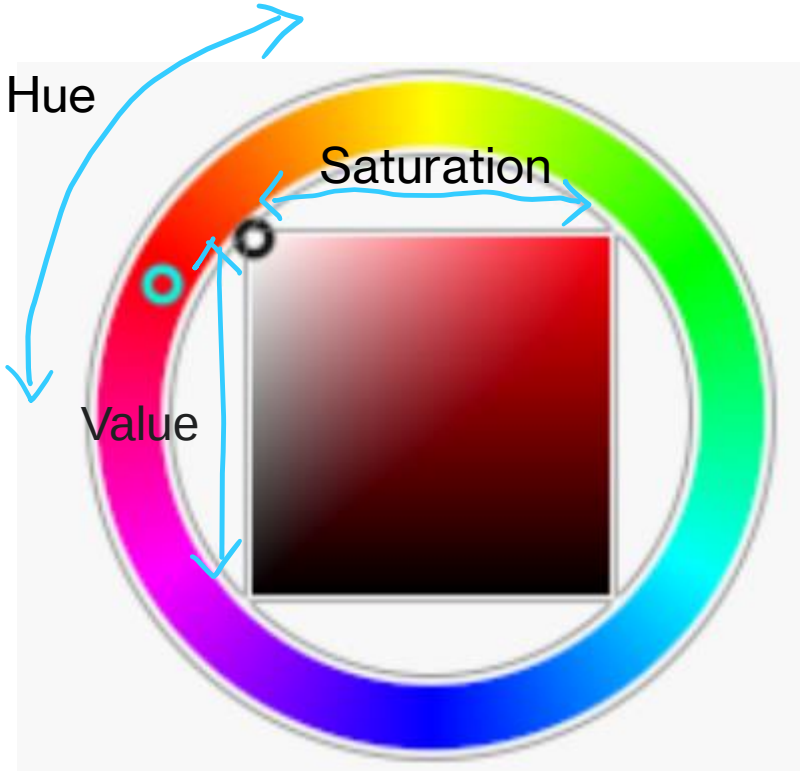
Related work



grayscale

color

1. Use black + white + gray to represent shadow & highlight
2. Color the object



Hue

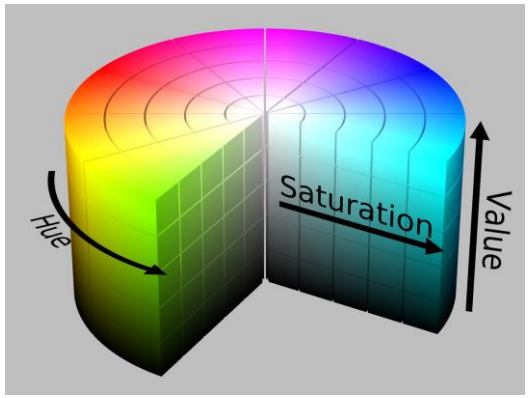
The "attribute of a visual sensation according to which an area appears to be similar to one of the perceived colors: red, yellow, green, and blue, or to a combination of two of them".

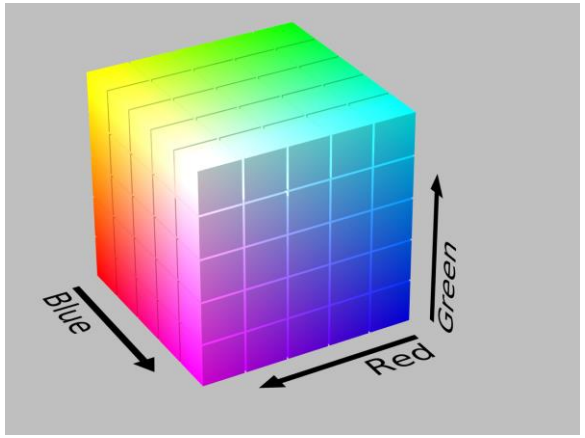
Saturation

The "colorfulness of a stimulus relative to its own brightness".

Value

The mixture of those paints with varying amounts of black or white paint





Conversion between RGB and HSV

From RGB [edit]

See also: § General approach

This is a reiteration of the previous conversion.

Value must be in range $R, G, B \in [0, 1]$.

With maximum component (i. e. value)

$$X_{max} := \max(R, G, B) =: V$$

and minimum component

$$X_{min} := \min(R, G, B) = V - C,$$

range (i. e. chroma)

$$C := X_{max} - X_{min} = 2(V - L)$$

and mid-range (i. e. lightness)

$$L := \operatorname{mid}(R, G, B) = \frac{X_{max} + X_{min}}{2} = V - \frac{C}{2},$$

we get common hue:

$$H := \begin{cases} 0, & \text{if } C = 0 \\ 60^\circ \cdot \left(0 + \frac{G - B}{C}\right), & \text{if } V = R \\ 60^\circ \cdot \left(2 + \frac{B - R}{C}\right), & \text{if } V = G \\ 60^\circ \cdot \left(4 + \frac{R - G}{C}\right), & \text{if } V = B \end{cases}$$

and distinct saturations:

$$S_V := \begin{cases} 0, & \text{if } V = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases}$$

$$S_L := \begin{cases} 0, & \text{if } L = 0 \text{ or } L = 1 \\ \frac{C}{1 - |2V - C - 1|} = \frac{2(V - L)}{1 - |2L - 1|} = \frac{V - L}{\min(L, 1 - L)}, & \text{otherwise} \end{cases}$$

Conversion between RGB and HSV

HSV to RGB alternative [edit]

Given a color with hue $H \in [0°, 360°]$, saturation $S = S_V \in [0, 1]$, and value $V \in [0, 1]$, first we define function :

$$f(n) = V - VS \max(0, \min(k, 4 - k, 1))$$

where $k, n \in \mathbb{R}_{\geq 0}$ and:

$$k = (n + \frac{H}{60°}) \bmod 6$$

And output R,G,B values (from $[0, 1]^3$) are:

$$(R, G, B) = (f(5), f(3), f(1))$$

Above alternative equivalent formulas allow shorter implementation. In above formulas the $a \bmod b$ returns also fractional part of module e.g. the formula $7.4 \bmod 6 = 1.4$. The values of $k \in \mathbb{R} \wedge k \in [0, 6)$. The base shape

$$t(n, H) = T(k) = \max(0, \min(k, 4 - k, 1))$$

is constructed as follows: $t_1 = \min(k, 4 - k)$ is "triangle" for which non-negative values starts from k=0, highest point at k=2 and "ends" at k=4, then we change values bigger than one to one by $t_2 = \min(t_1, 1) = \min(k, 4 - k, 1)$, then change negative values to zero by $t = \max(t2, 0)$ – and we get (for $n = 0$) something similar to green shape from Fig. 24 (which max value is 1 and min value is 0). The R,G,B functions of H use this shape transformed in following way: modulo-shifted on X (by n) (differently for R,G,B) scaled on Y (by $-VS$) and shifted on Y (by V). We observe following shape properties(Fig. 24 can help to get intuition about this):

$$t(n, H) = 1 - t(n + 3, H)$$
$$\min(t(n, H), t(n + 2, H), t(n + 4, H)) = 0$$
$$\max(t(n, H), t(n + 2, H), t(n + 4, H)) = 1$$

Several things seem worth noticing already:

- Most of the complexity comes from the hue calculation.
- Four min/max operations are performed to find `rgb_max` and `rgb_min`; however, sorting three values can be done with only 3 comparisons. This is not necessarily problematic because min/max could be wired in an efficient way depending on the CPU.
- Two additional tests are performed to compare `r` and `g` to `rgb_max`; if `rgb_max` and `rgb_min` were computed using tests, this is a waste of time to compare them again.
- Adding 6.f to the final hue value only has a 16.6% chance of happening.

The actual hue calculation depends on how *r*, *g*, and *b* are ordered:

$$\text{Hue}_{0\ldots6}(r,g,b) = \begin{cases} (g-b)/(r-b), & \text{if } r \geq g \geq b. \\ 6+(g-b)/(r-g), & \text{if } r \geq b \geq g. \\ 2+(b-r)/(g-r), & \text{if } g \geq b \geq r. \\ 2+(b-r)/(g-b), & \text{if } g \geq r \geq b. \\ 4+(r-g)/(b-g), & \text{if } b \geq r \geq g. \\ 4+(r-g)/(b-r), & \text{if } b \geq g \geq r. \end{cases}$$

But let's rewrite this in terms of *x*, *y* and *z*, where *x* is the largest of *(r,g,b)*, *z* is the smallest of the three, and *y* is inbetween:

$$\text{Hue}_{0\ldots6}(R,G,B) = \begin{cases} (y-z)/(x-z), & \text{if } r \geq g \geq b. \\ 6+(z-y)/(x-z), & \text{if } r \geq b \geq g. \\ 2+(y-z)/(x-z), & \text{if } g \geq b \geq r. \\ 2+(z-y)/(x-z), & \text{if } g \geq r \geq b. \\ 4+(y-z)/(x-z), & \text{if } b \geq r \geq g. \\ 4+(z-y)/(x-z), & \text{if } b \geq b \geq r. \end{cases}$$

There are a lot of similarities here. We can push it even further, using the fact that $x \geq z$ and $y \geq z$ by definition:

$$\text{Hue}_{0\ldots6}(R,G,B) = \left| K + \frac{y-z}{x-z} \right|, \text{with } K = \begin{cases} 0, & \text{if } r \geq g \geq b. \\ -6, & \text{if } r \geq b \geq g. \\ 2, & \text{if } g \geq b \geq r. \\ -2, & \text{if } g \geq r \geq b. \\ 4, & \text{if } b \geq r \geq g. \\ -4, & \text{if } b \geq b \geq r. \end{cases}$$

That's actually the same calculation! Only the hue offset *K* changes. The idea now is the following:

- Sort the triplet *(r,g,b)* using comparisons
- Build *K* while sorting the triplet
- Perform the final calculation

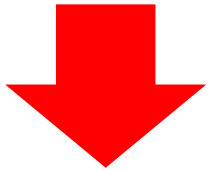
http://lolengine.net/blog/2013/01/13/fast-rgb-to-hsv

```glsl
vec3 rgbTohsv(vec3 c)
{
    vec4 K = vec4(0.0, -1.0 / 3.0, 2.0 / 3.0, -1.0);
    vec4 p = mix(vec4(c.bg, K.wz), vec4(c.gb, K.xy), step(c.b, c.g));
    vec4 q = mix(vec4(p.xyw, c.r), vec4(c.r, p.yzx), step(p.x, c.r));

    float d = q.x - min(q.w, q.y);
    float e = 1.0e-10;
    return vec3(abs(q.z + (q.w - q.y) / (6.0 * d + e)), d / (q.x + e), q.x);
}

vec3 hsvTorgb(vec3 c)
{
    vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
    vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
    return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
}
```

http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl

From what we have learnt in NPR Tone-based Illumination + Shading, object colors change in sunlight scenes because cool skylight and warm sunlight vary in relative contribution across the surface.

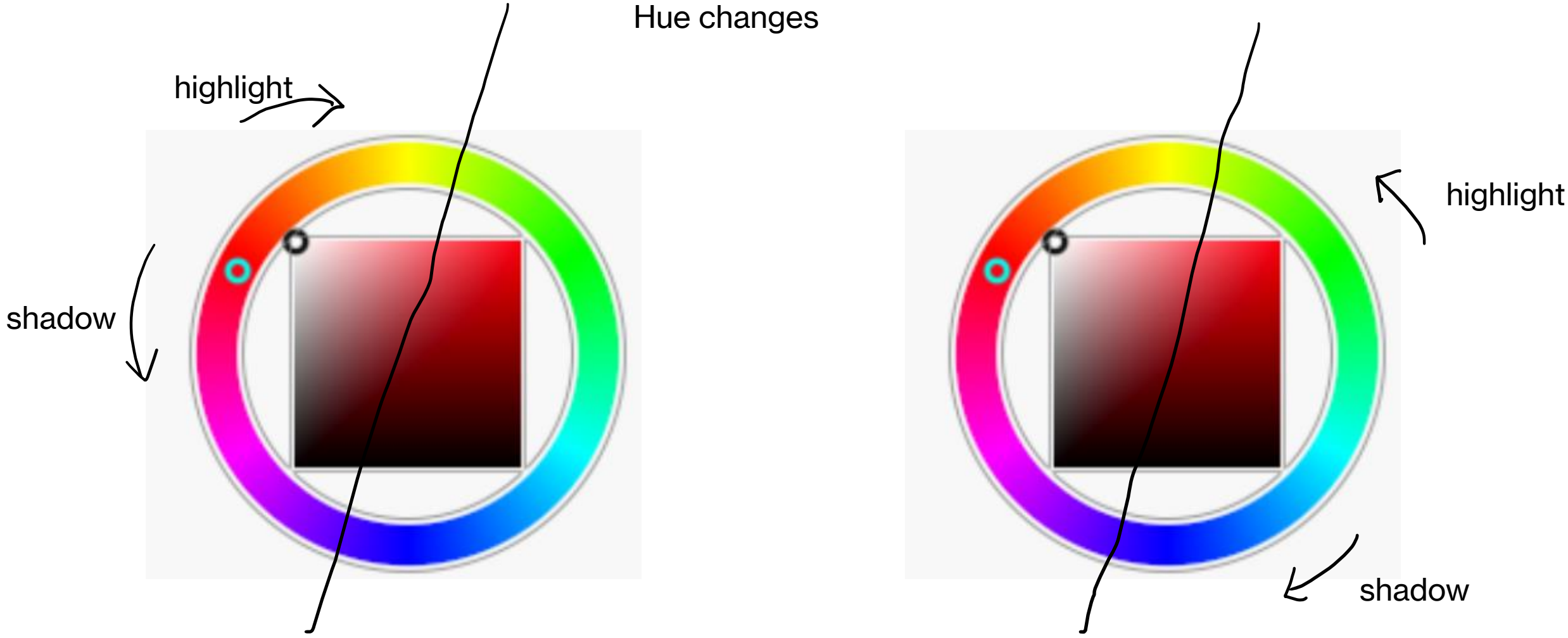
From what we have learnt in NPR Tone-based Illumination + Shading, object colors change in sunlight scenes because cool skylight and warm sunlight vary in relative contribution across the surface.



Hue changes

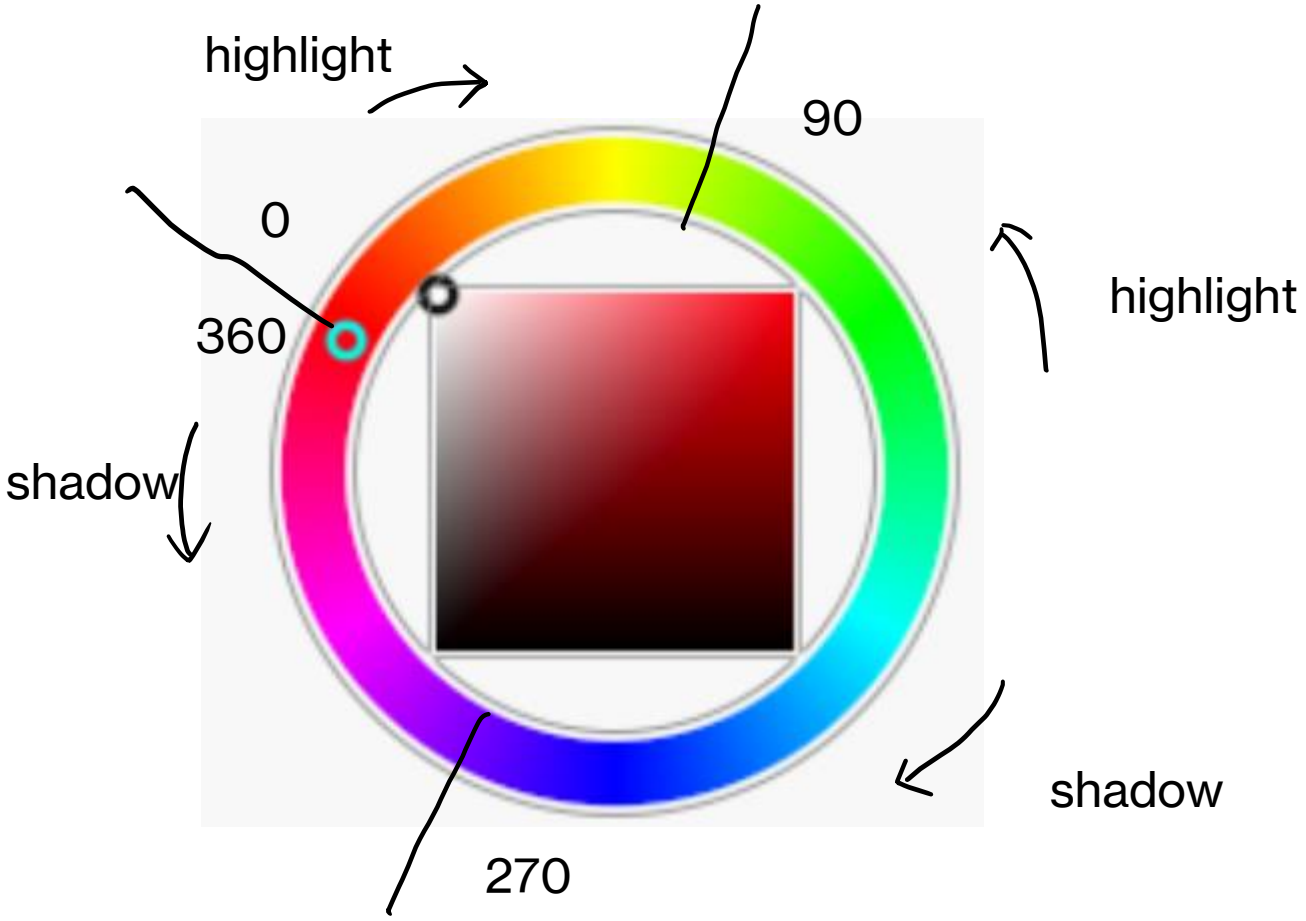
From what we have learnt in NPR Tone-based Illumination + Shading, in real world, object colors change in sunlight scenes because cool skylight and warm sunlight vary in relative contribution across the surface.

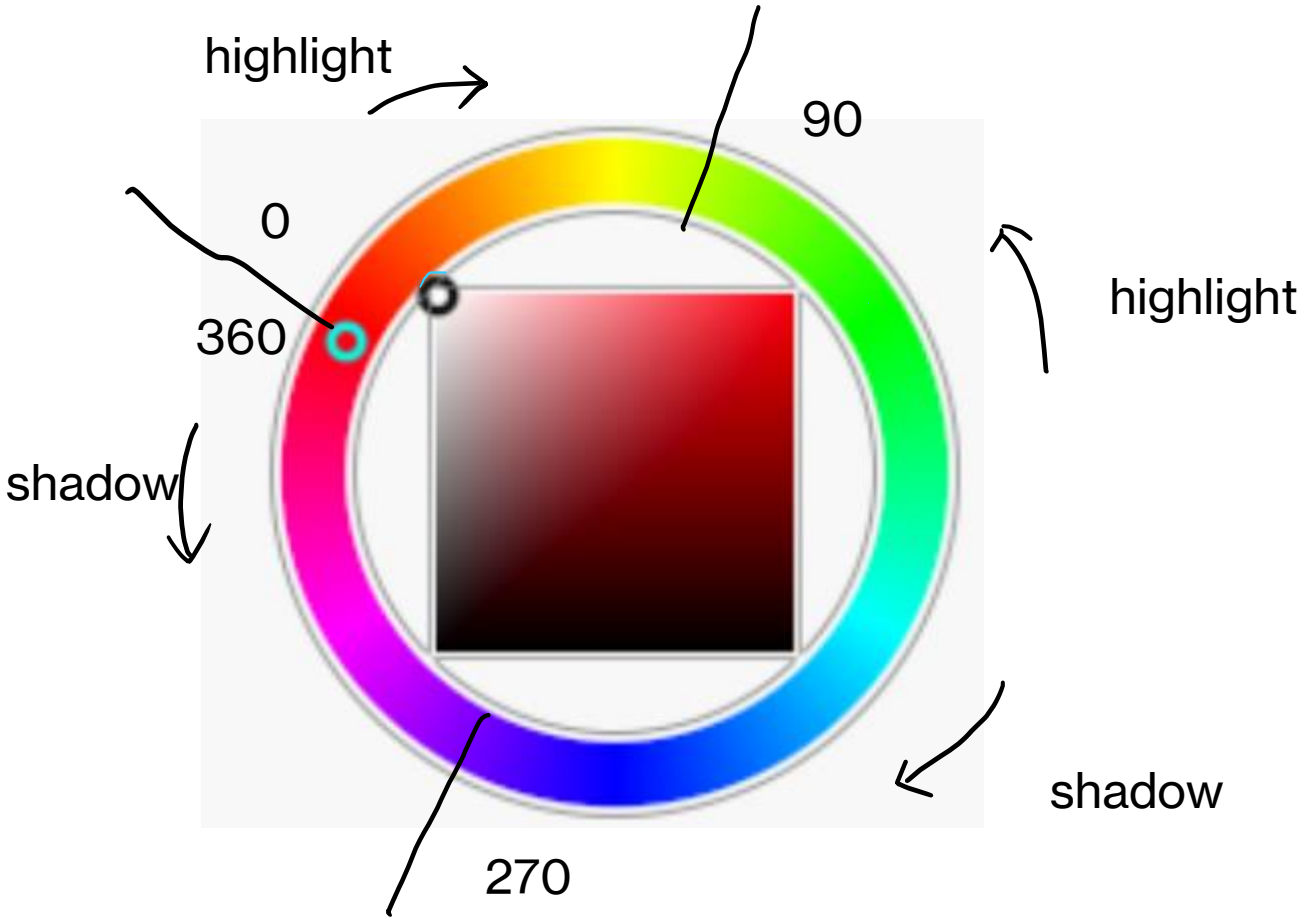
Hue changes



Near-silhouette

+ color

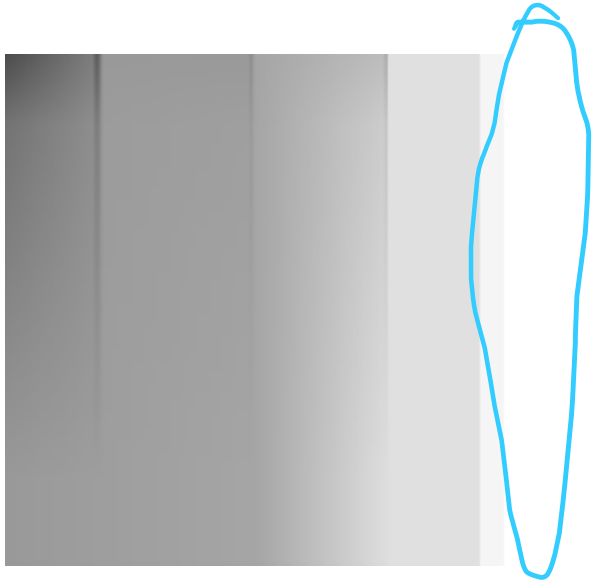

highlight
90
0
360
highlight
shadow
shadow
270




```
if(hsvColor.x > 0.25 && hsvColor.x <= 0.75){
     if(s >= 0.5){
        float val = s - 0.5;
        hsvColor.x = hsvColor.x  - val * hVal;
    }
    else{
        float val = 0.5 - s;
        hsvColor.x = val * hVal + hsvColor.x;

    }
}
else{
     if(s >= 0.5){
        float val = s - 0.5;
        hsvColor.x = val * hVal + hsvColor.x;

    }
    else{
        float val = 0.5 - s;
        hsvColor.x = hsvColor.x  - val * hVal;
    }
}
```

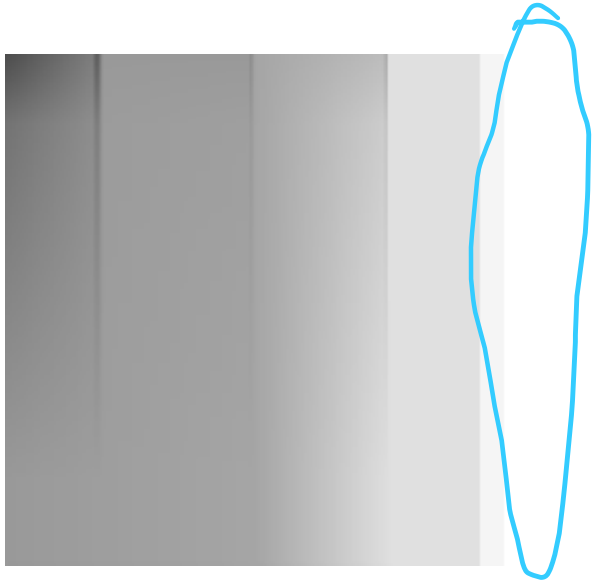
Highlight

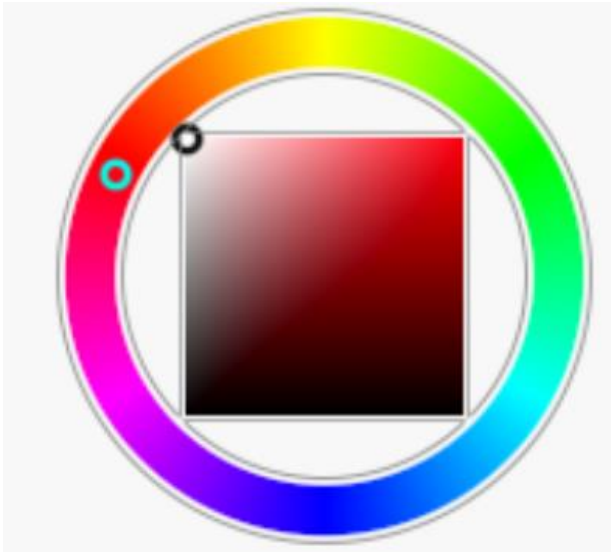
white + other color = other color



Highlight

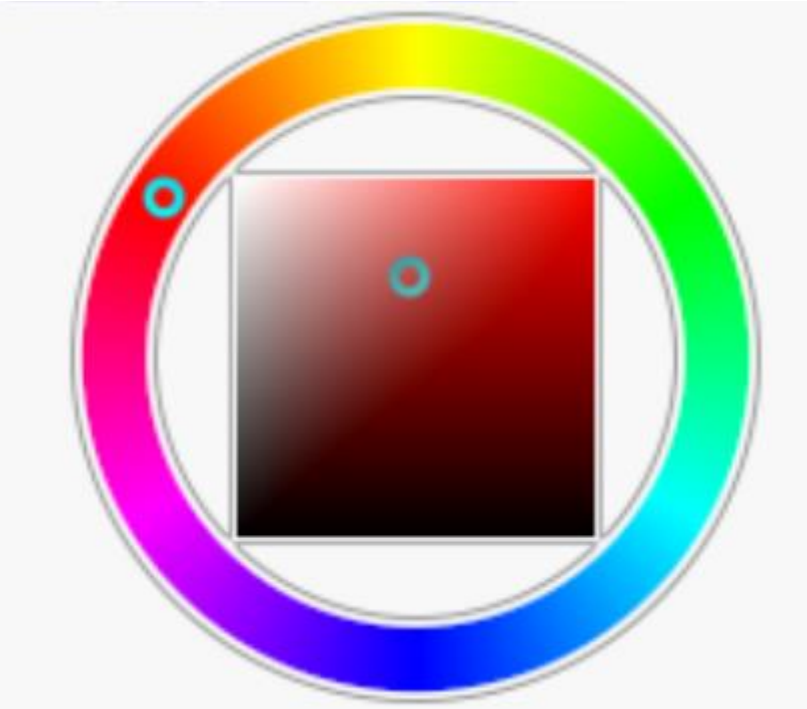
white + other color = other color

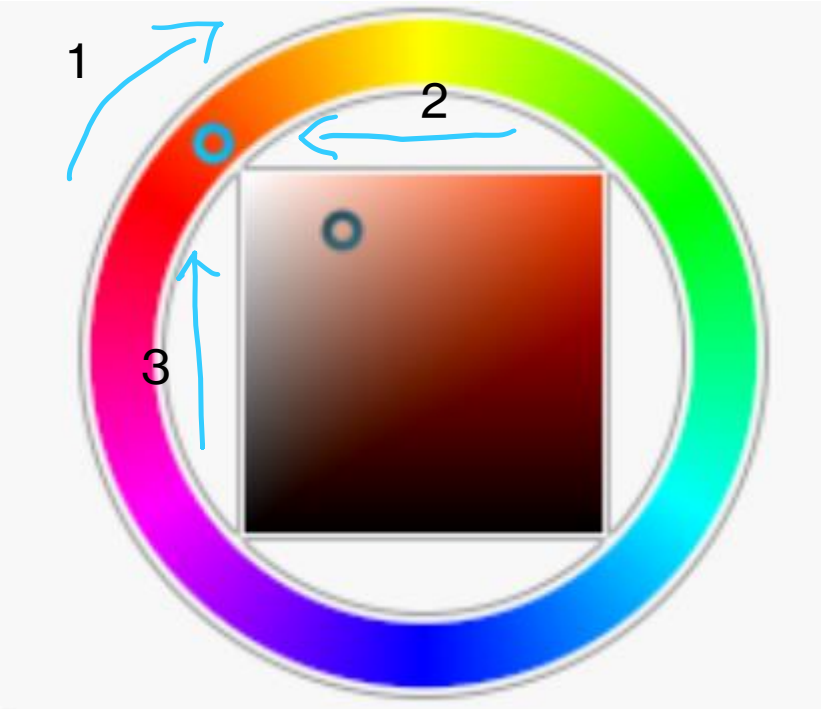




Highlight

white + other color = other color



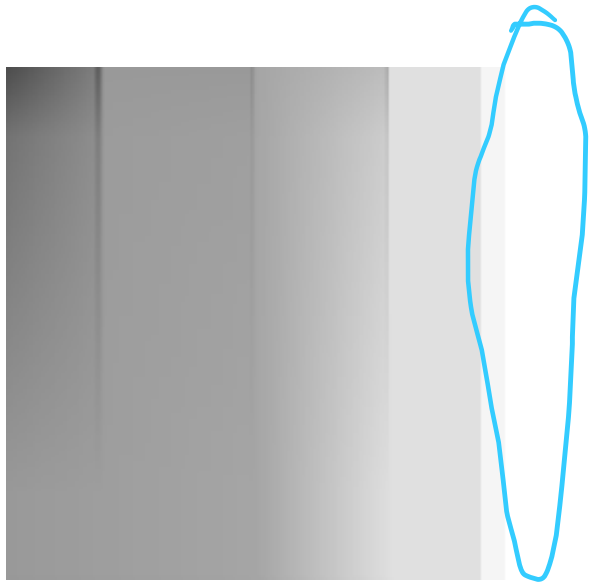


1. Change H value
2. Decrease S value
3. Increase V value

Highlight

white + other color = other color

```
if(s > 0.94){
    hsvColor.y = log(1 + t) * hsvColor.y;
    hsvColor.z = min(log(1 + s) + s, 1) * hsvColor.z;
}
```



Final color

Original

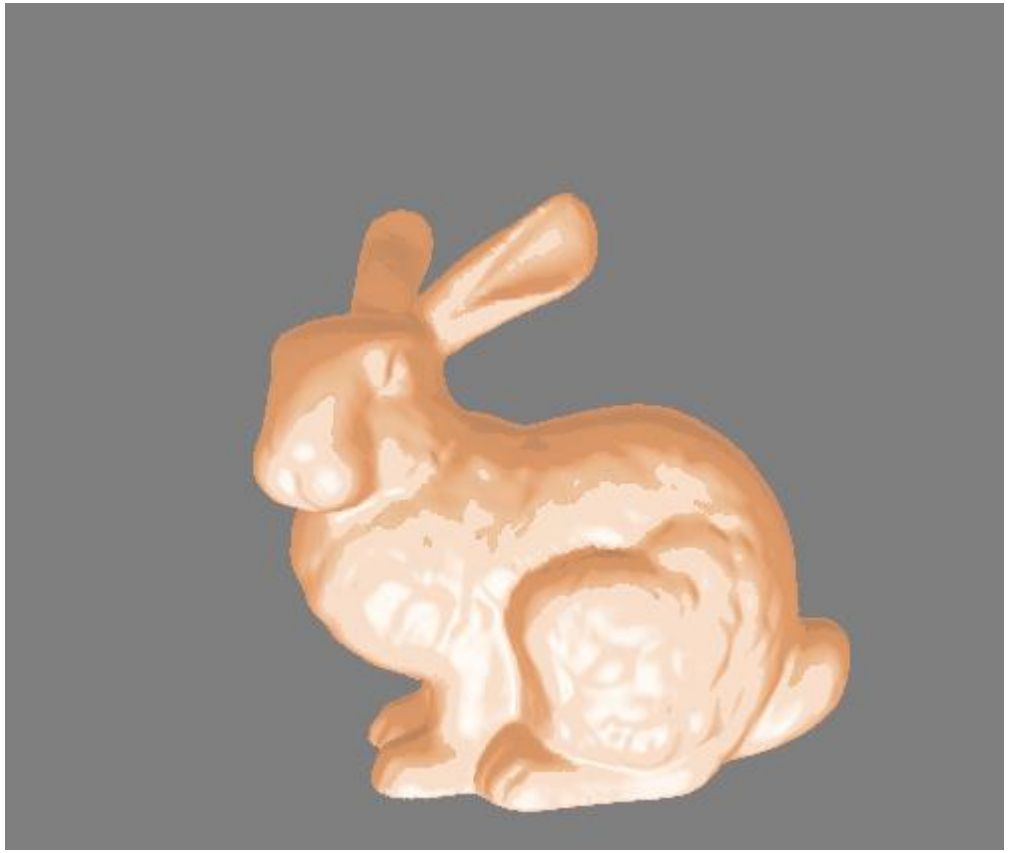
Changed hVal

Changed opacity(how much to blend the texture and color)

```
FragColour = vec4(texColour.xyz * objColor * opacity + texColour.xyz * (1 - opacity), 1.0);
```

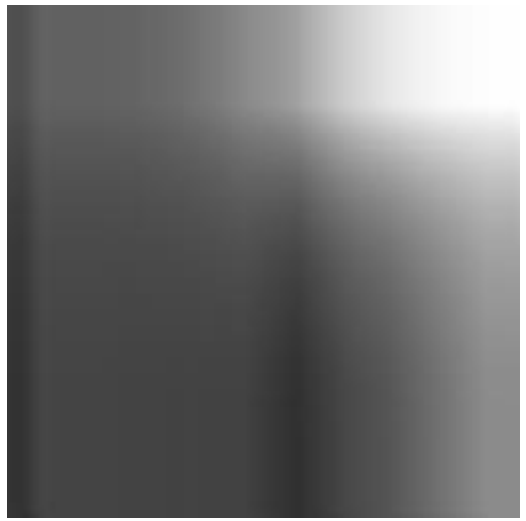
Compared with assignment 3

project



assignment

Metal

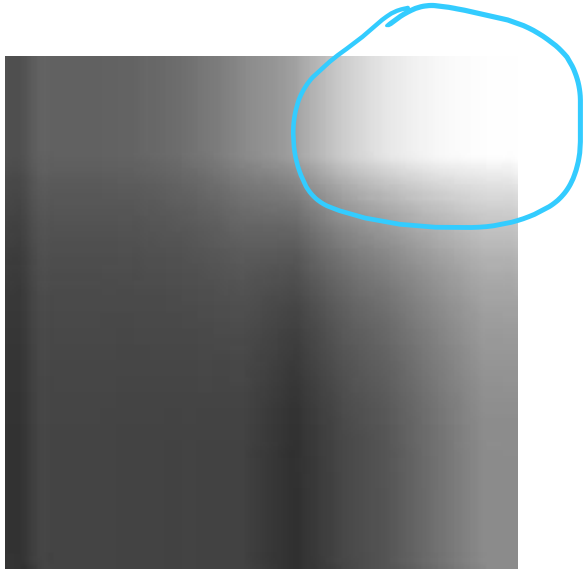


+ color

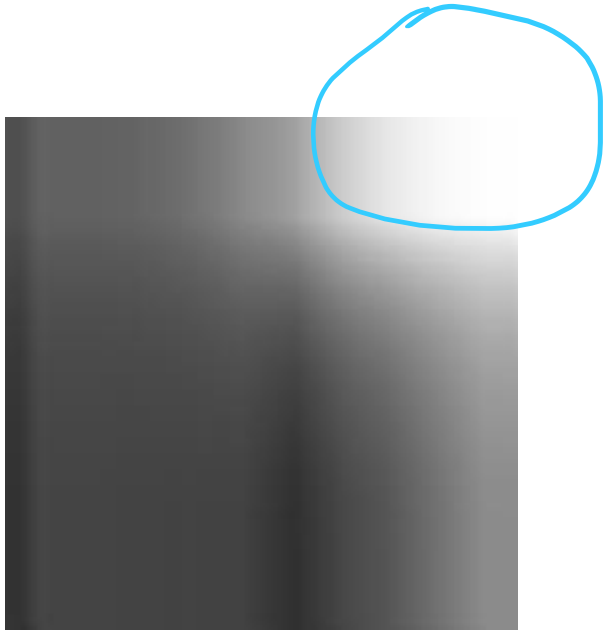
Hue is the same as near-silhouette texture
But metal has shiny highlight

Blend

white + other color = other color



Hue is the same as non-metal texture
But metal has shiny highlight



```
//if it's the highlight
if(t >= 0.79 && s >= 0.64){
    hsvColor.y = (1 - t) * hsvColor.y;
    hsvColor.z = (1 + s) /2 * hsvColor.z;
}
```

Final color

Original

Changed hVal

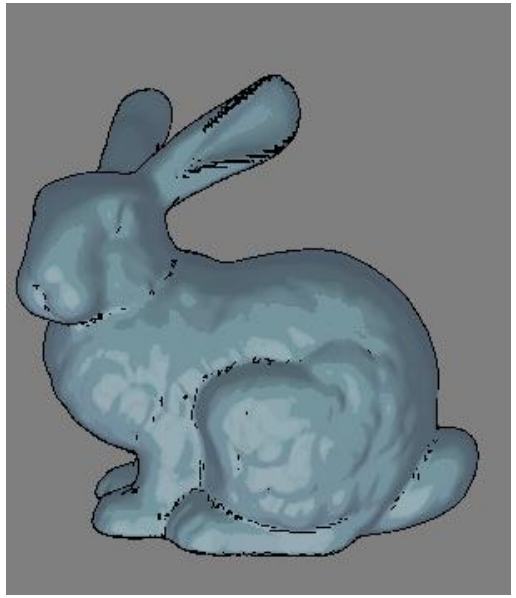
Changed opacity(how much to blend the texture and color)

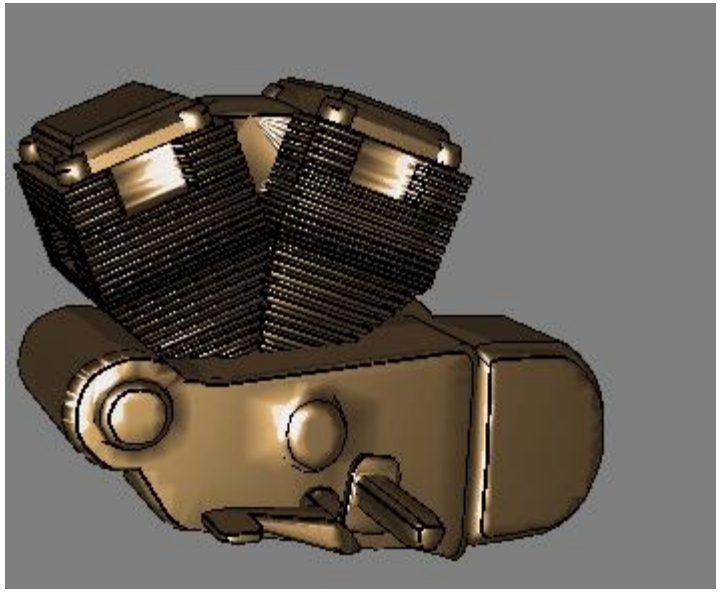
FragColour = vec4(texColour.xyz * objColor * opacity + texColour.xyz * (1 - opacity), 1.0);

Compared with assignment 3

project

assignment





Result

Limitations & future directions

Explore more textures
Improve feature lines
Use function model to generate textures

Reference

1. Gooch-et-al-1998
2. Buchanan-Costa_Sousa-2000
3. Barla-et-al-2006
4. Maureen C. Stone (August 2001). "A Survey of Color for Computer Graphics". Course at SIGGRAPH 2001.
5. MacEvoy, Bruce. "handprint : colormaking attributes". *www.handprint.com*. Retrieved 26 February 2019.
6. https://en.wikipedia.org/wiki/HSL_and_HSV#Formal_derivation
7. http://lolengine.net/blog/2013/01/13/fast-rgb-to-hsv
8. http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl