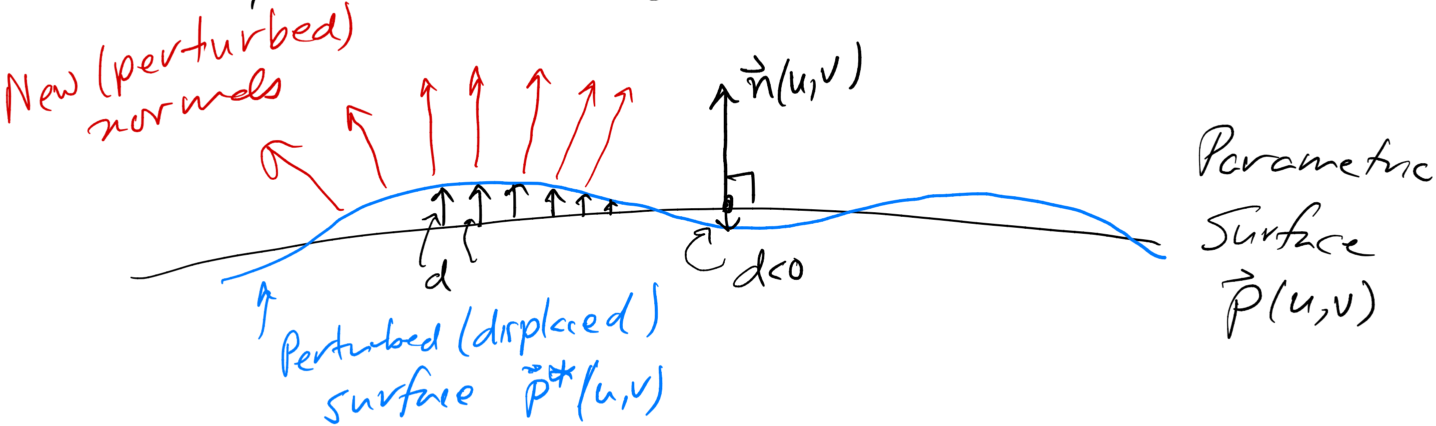
Bump mapping:

A method of "faking" the normals of a surface, so that Phong lighting gives the appearance of a bumpy.

Only works with Phong interpolation



Displacements – in direction of normals $\vec{n}(u,v)$, distance $d(u,v)$

$$\vec{p}^*(u,v) = \vec{p}(u,v) + d(u,v)\cdot\vec{n}(u,v) = \boxed{\vec{p}^* = \vec{p} + d\vec{n}}$$

$$\vec{p}^* = \vec{p} + d \cdot \vec{n}$$

$$\vec{n} = \frac{\partial \vec{p}}{\partial u} \times \frac{\partial \vec{p}}{\partial v}$$ (Non normalized)

$$\vec{n} = \vec{p}_u \times \vec{p}_v$$

Perturbed normal

$$\vec{n}^* = \vec{p}^*_u \times \vec{p}^*_v$$

$$\frac{\partial p^*}{\partial u} = \vec{p}^*_u = \vec{p}_u + d_u \cdot \vec{n} + d \cdot \vec{n}_u \approx 0$$

$$\vec{p}^*_u \approx \vec{p}_u + d_u \vec{n} = \frac{\partial \vec{p}}{\partial u} + \frac{\partial d}{\partial u} \cdot \vec{n}$$

Likewise $\vec{p}^*_v \approx \vec{p}_v + d_v \vec{n}$

So $\vec{n}^* = \vec{p}_u \times \vec{p}_v \approx \boxed{\vec{p}_u \times \vec{p}_v + d_u \vec{n} \times \vec{p}_v + d_v \vec{p}_u \times \vec{n}}$

Need to know: $\vec{p}_u, \vec{p}_v, d_u, d_v, \vec{n}$

d - displacement - can be given in a texture map.

$\frac{\partial d}{\partial u}, \frac{\partial d}{\partial v}$ - can compute with finite differences. Alternately: the Texture Map holds d_u, d_v directly.

Color: RGB / CMY(K) - not covering HSL or sRGB.

(First part of the Color Chapter).

Trichromatic theory of Color

(Most) humans see colors in 3 categories Red, Green, Blue.

Palmer 1777
Young 1801

Physiology Retina has

3 kinds of cones
selectively sensitive to Red, Green, Blue

and Rods - sensitive to light at low light levels, but don't discern ~~not~~ separate colors

Opponent Theory of Vision - Hering 1878

(Most) humans see color in

Red vs Green

Blue vs Yellow

Light vs Dark.

Physiology: Your retina encodes colors into these three channels

Both theories   Light comes in a 3D space
at standard levels of illumination

Any color visible color "C" can be expressed as

$$C = \alpha R + \beta G + \gamma B \qquad \alpha + \beta + \gamma = 1$$

Sometimes α, or β or γ < 0 (negative)

Additive Colors

Red, Green, Blue

-example a monitors, or this screen

Start with a black background,

add red, green, blue light in mixtures.

Subtractive colors

Start with white & subtract out

Cyan, Magenta or Yellow

Example - film, paint, inks, printed ~~matter~~

Printers CMYK K - black.

Professional Printers - Use CMYK + OGV — violet

green

orange

Nominal conversion between RGB and CMY

is

$$R = 1 - C \qquad C = 1 - R$$
$$G = 1 - M \qquad M = 1 - G$$
$$B = 1 - Y \qquad Y = 1 - B$$

Color representations

OpenGL often: floating point for RGB values.

Common representation: 8-bits for R, G, B each

called 24-bit or 32-bit color.

"millions of colors"

32-bit: RGBA

A - "alpha" & transparency.

Old representation: 16 bit color.

"thousands of colors"

5 bits for R, G, B, 1-bit for A.

8-bit color:
3 bits for Red
3 " " Green
2 bits for Blue

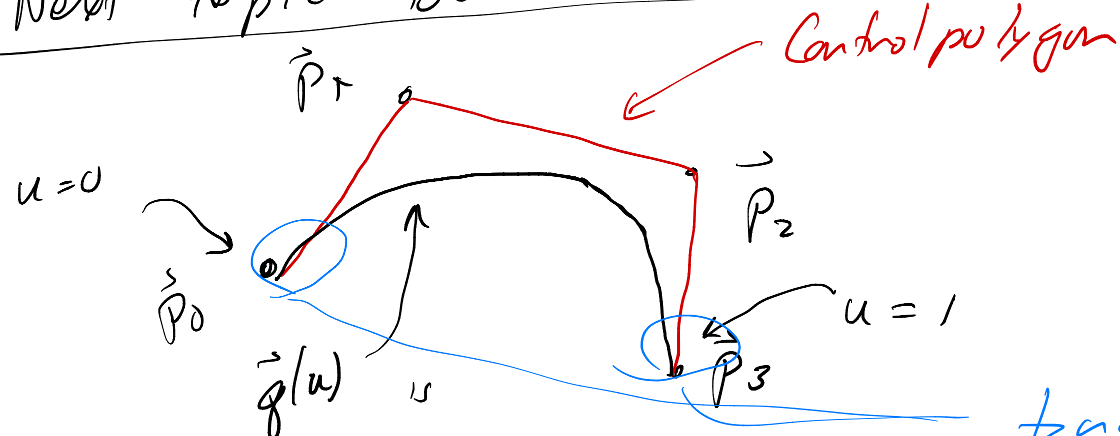
CLUT - Color look-up table

Typically - table of 256 many 24 (or 32) bit colors
Each pixel in an image has a 8-bit byte specifying
of the 256 colors.

GIF's use this. (Plus they use Lempel-Ziv compressions).

Rec 2020 for UHDTV
uses either 10 or 12 bits for each of Red, Green, Blue.

Next topic: Bézier curves



$\vec{p}_0, \vec{p}_1, \vec{p}_2, \vec{p}_3$ in $\mathbb{R}^?$

tangent to the control polygon at the start & end points

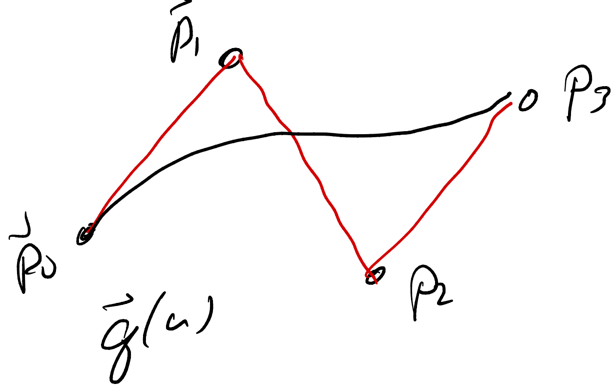
$\vec{q}(u)$ is a Bézier curve

Defined for $u \in [0,1]$

$\vec{q}(0) = \vec{p}_0 \qquad \vec{q}(1) = \vec{p}_3$

$\vec{q}\,'(0) = 3(\vec{p}_1 - \vec{p}_0) \qquad \vec{q}\,'(1) = 3(\vec{p}_3 - \vec{p}_2)$

Control entirely of $\vec{q}(u)$ by just 4 points $\vec{p}_0, \vec{p}_1, \vec{p}_2, \vec{p}_3$



Degree 3
Bézier curve

Font design: - Boundaries of letters are Bézier curves.

Manufacturing: