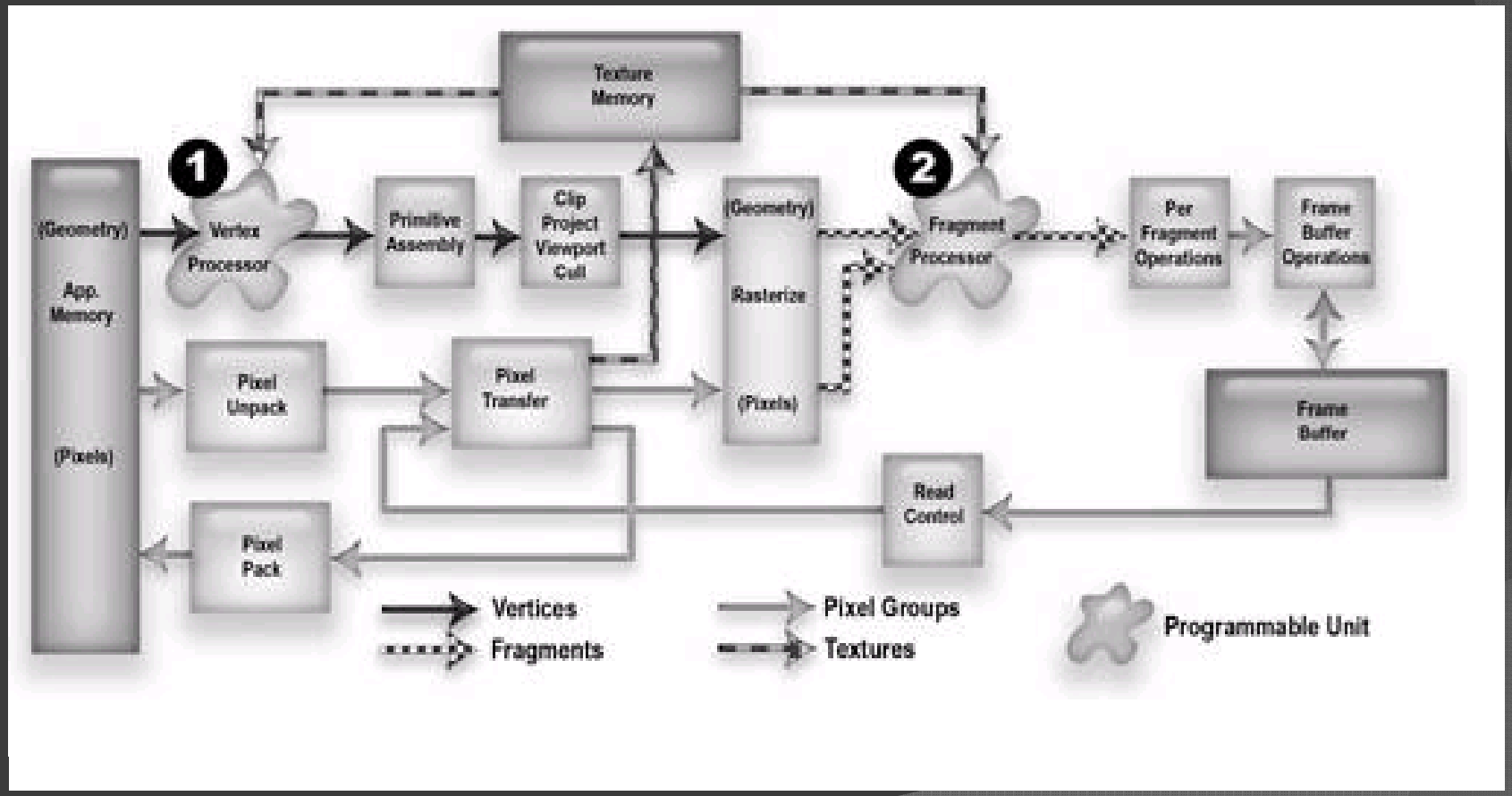
GLSL- OpenGL Shading Language

- Jezik je dodan kao dio OpenGL standarda od verzije 2.0
- Uvodi se mogučnost programirljivosti u ključnim fazama procesiranja geometrije.
- Uz pomoć GLSL-a, faze fiksnog protoka za vertex i fragment procesiranje, zamjenjene su s programirljivim fazama koje mogu raditi sve poslove fiksnog protoka i više.
- GLSL kod koji je namjenjen izvršavanju na programirljivom procesoru (GPU) nazivamo shader.
- Pošto su dva programirljiva procesora definirana razlikujemo Vertex Shader i Fragment Shader (dodan je još i Geometry Shader)
- Jezik je baziran na sintaksi i kontroli toka C i C++
- Isti jezik uz mali skup promjena se koristi i u Vertex i u Fragment Shader-u
- Ovime se postiže sloboda definiranja procesiranja koja će se dogodit.

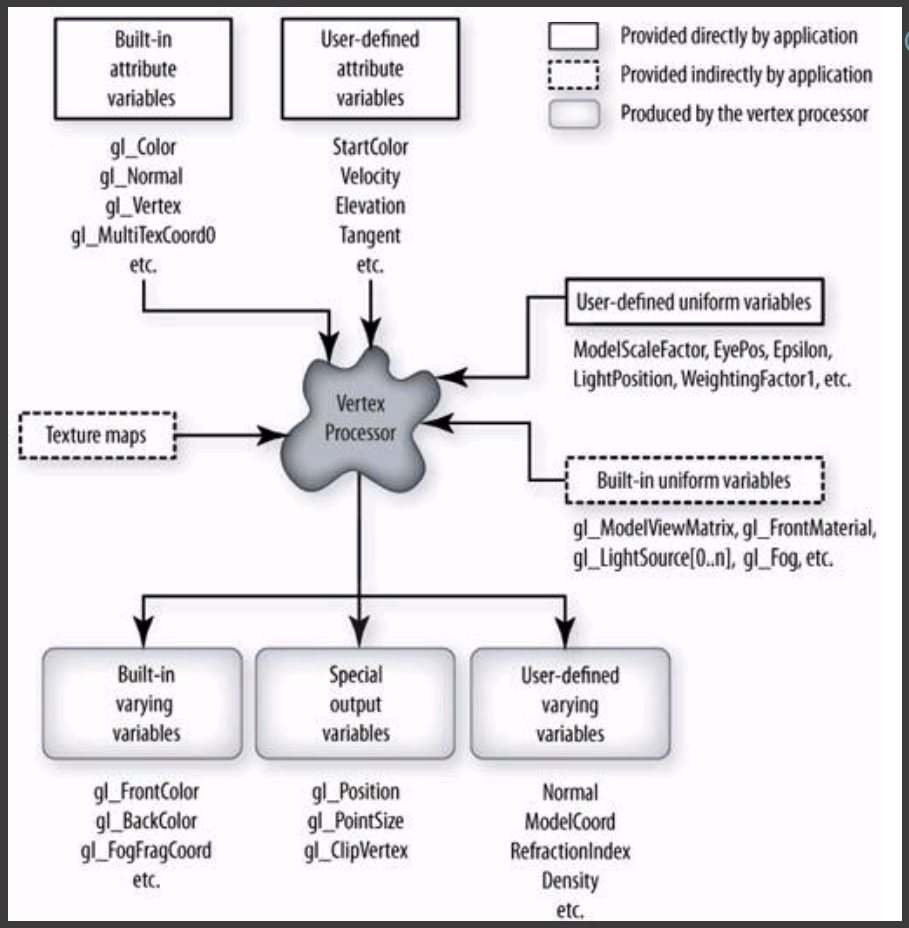
OpenGL logički dijagram koji pokazuje programirljivr procesore za Vertex i Fragment Shader-e



Vertex Procesor

- Vertex procesor je programirljiva jedinica koja vrši operacije nad ulaznim vertex vrjednostima.
- Obično vrši sljedeće operacije: Transformacije, transformacije normale i normalizacije, generiranje koordinata tekstura, proračun svjetala, primjena boja materijala. No, zbog svoje generalne programirljivosti na njemu se mogu računati razni drugi proračuni ovisno o potrebi programera.
- Vertex procesor ima svoje ulaze pomoću kojih postavlja izlaze.
- Razlikujemo ugrađene i korisnički postavljene ulaze i izlaze
- Nije moguće npr. da postojeći fiksni tok obavlja operacije transformacije pozicije i normale, a da vertex procesor obavlja proračun svjetlja.Vertex procesor mora obavljat svo troje, jer on u potpunosti zamjenuje tu fazu fikdnog toka.

Unutar Vertex procesora varijable mogu imati tri različita kvalifikatora:
- attribut varijable(predstavljaju varijable koje se mjenjaju na razini vertex-a)
- uniform varijable(predstavljaju varijable koje se mjenjaju na razini primitive)
- varying varijable(predstavljaju vrjednosti koje vertex šalje fragment procesoru)

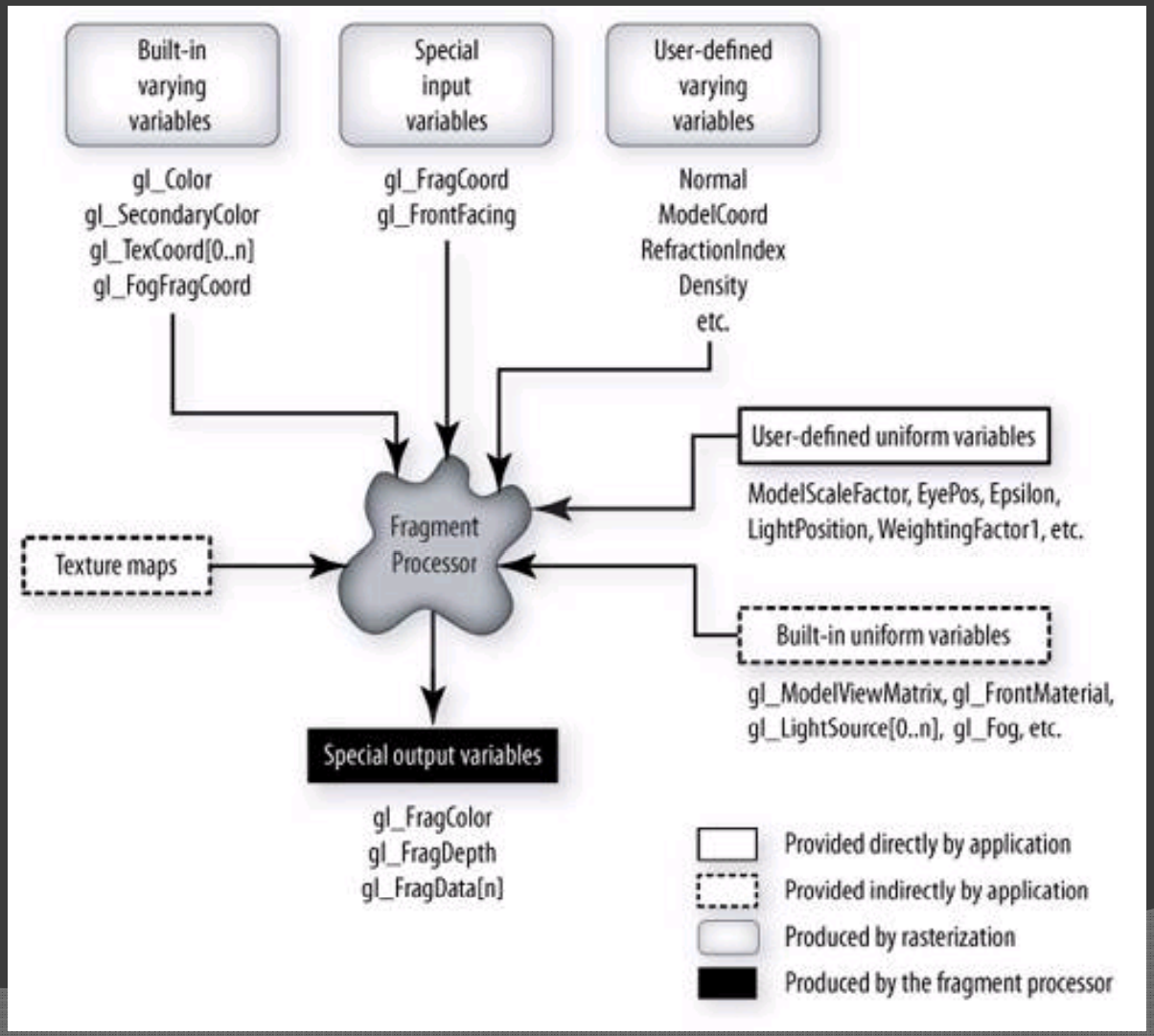


- Attribute varijable se odnose samo na vertex procesor, dok uniform varijable prenose vrjednosti iz aplikacije u fragment ili vertex procesor(jedna uniform varijabla je globalna unutar para vertex i fragment shader-a)

Fragment Procesor

- Fragment procesor je programirljiva jedinica koja vrši operacije nad fragment vrjednostima i njima asociranim podacima
- Obično vrši sljedeće operacije: Operacije nad interpoliranim vrjednostima, pristup i primjena tekstura, magla i računanje konačne boje.
- Fragment shader ne može promjenit x i y poziciju fragmentu.
- Fragment shader ne zamjenuje operacije fiksnog protoka koje se nalaze na samom kraju pixel protočnog sustava(scissor test, alpha test, depth test...)
- Primarni ulazi fragment procesora su interpolirane varying varijable koje su prenesene iz vertex procesora

Za svaki fragment fragment procesor može izračunati boju, dubinu i proizvoljne vrjednosti(pisanje u varijable gl_FragColor, gl_FragDepth, gl_FragData) ili u potpunosti odbaciti fragment



Jednostavni primjer:

```
uniform float CoolestTemp;
uniform float TempRange;
attribute float VertexTemp;

varying float temp;

void main()
{
	temp = (VertexTemp - CoolestTemp);
	temp /= TempRange;
	gl_Position = ftransform();
//gl_Position = gl_ModelViewProjectionMatrix
	*  gl_Vertex
}
```

Vertex Shader

```
uniform  vec3 HottestColor;
uniform  vec3 CoolestColor;

Varying float temp;

void main()
{
	vec3 color = mix(CoolestColor,
	HottestColor, temp);

	gl_FragColor = vec4(color, 1.0);
}
```

Fragment Shader

Textutre Mapping

- Texture mapping nam omogučuje priljepljivanje slike na poligon i iscrtavanje takvog poligona. Teksture su najjednostavnije rečeno polje podataka.Element teksture nazivamo texel.
- Tekstura može biti definirana kao 1D, 2D, 3D,...
- Koraci za primjenu texture mapping-a:
  - Pokazat na koji način se tekstura primjenjuje za svaki pixel
  - Omogučavanje texture mapping-a
  - Iscrtati scenu, davajući i koordinate teksture i geometrije

Jedna od mogučnosti primjene teksture na fragment bi bila da se boja teksture koristi kao krajna boja fragmenta(replace).Druga metoda bi bila da se boja teksture koristi za skaliranje fragmenta(modulate).Konstantna boja može biti blendana sa bojom fragmenta s obzirom na vrjednost teksture.

Bump Mapping

- Bump mapping modulira normale površine prije nego je proračun svijetla primjenjen.Bump mapping povečava realizam objekta, bez da povečava kompleksnost geometrije.
- Tehnika u načelu ne mijenja normale, već na neki način prevari proračun svijetla.

Običan texture mapping

Bump Mapping

Klasična formulacija bump mapping koju je razvio Blinn, računa izmjenjene normale površine kao da je funkcija visine nadodana na ravnu površinu u smjeru ispravne normale.

Neka P predstavlja točku na parametriziranoj površini , računamo normalu u toj točci:

$$\mathbf{N}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial u} \times \frac{\partial \mathbf{P}(u,v)}{\partial v}$$

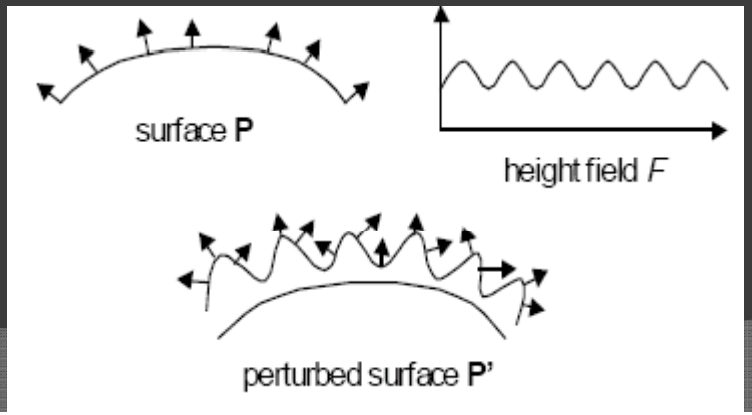
Promjenimo vektor pozicije površine, dodavajući mu bump funkciju:

$$\mathbf{P'}(u,v) = \mathbf{P}(u,v) + F(u,v)\frac{\mathbf{N}(u,v)}{|\mathbf{N}(u,v)|}$$

$$\mathbf{N'}(u,v) = \frac{\partial \mathbf{P'}(u,v)}{\partial u} \times \frac{\partial \mathbf{P'}(u,v)}{\partial v}$$

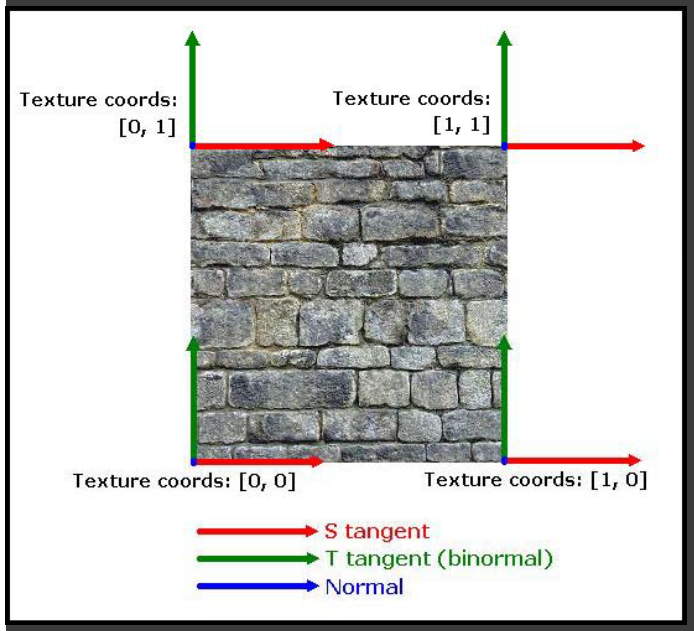
Normala točke definirane nad P':

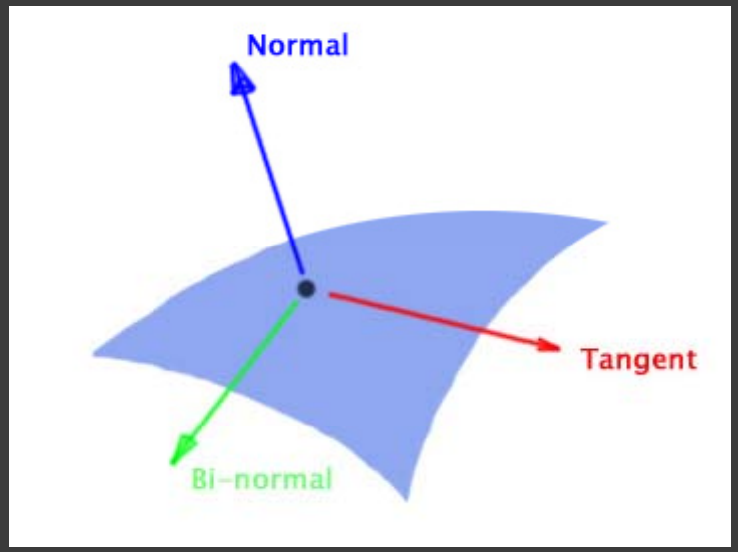
Uz računanje parcijalnih derivacija dobijemo:



$$\mathbf{N'} = \mathbf{N} + \frac{\frac{\partial F}{\partial u}\left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial v}\right) - \frac{\partial F}{\partial v}\left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial u}\right)}{|\mathbf{N}|}$$

Da bi rezultat proračuna svijetla imao smisla, svi vektori koji se koriste pri tom proračunu moraju biti definirani unutar istog koordinatnog sustava.
Kada bi za taj sustav odabrali očne koordinate, tada bi promjenjena nomala koju bi koristili za taj proračun morala bi ne samo biti prebačena u taj sustav, nego i dodana ispravnoj normali i normalizirana što su sve skupa jako skupe operacije



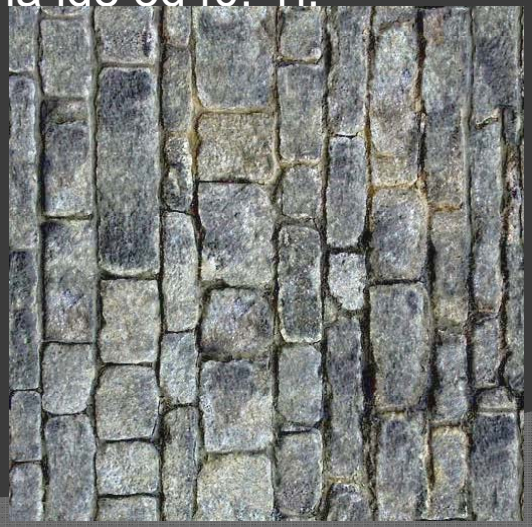


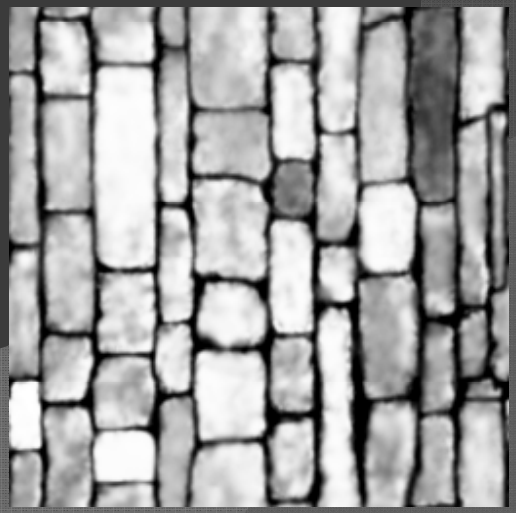
Kao koordinatni sustav odabrat ćemo sustav koji varira nad svakim vertex-om objekta. On predpostavlja da se svaka točka nalazi na poziciji (0, 0, 0) i da svaki nepromjenjeni vektor normale nalazi se u svakoj točci na poziciji (0, 0, 1).U ovakav sustav bi morali prebacit vektor pogleda i svijetla.Da bi dobili matricu transformcije koja će prebacivat vektore u taj sustav, zahtjevamo od aplikacije još jedan dodatni vektor, tangentu.Po definiciji taj tangent vektor je u ravnini površine i okomit na ulaznu normalu.Ako napravimo cross produkt između normale i tog tangent vektora dobivamo treći vektor kojeg zovemo binormala.Zajedno ova tri vektora čine ortonormalnu bazu ovog sustava.Pošto ovaj sustav definira tangent vektor kao svoj osnovni vektor ovaj sustav nazivamo Tangent koordinatni sustav

Transformiramo vektor (Ox, Oy, Oz) iz prostora svijeta u tangent prostor pomoću matrice rotacije T.

$$\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix}$$

U svrhu spremanja tih izmjenenih normala koje ćemo zapravo koristiti pri proračunu svjetla, definiramo normal mape. To su teksture koje u svakom svom texel-u kao RGB vrjednosti sadrže normale. Postavlja se problem kako ih računati i kako spremiti jer ipak boje unutar teksture su definirane od [0, 1]. Normale se računaju pomoću height mapa. height mape su teksture koje u svakom svom texel-u sadrže visinu za odgovarajuću poziciju u teksturi boje, ta visina ide od [0, 1].



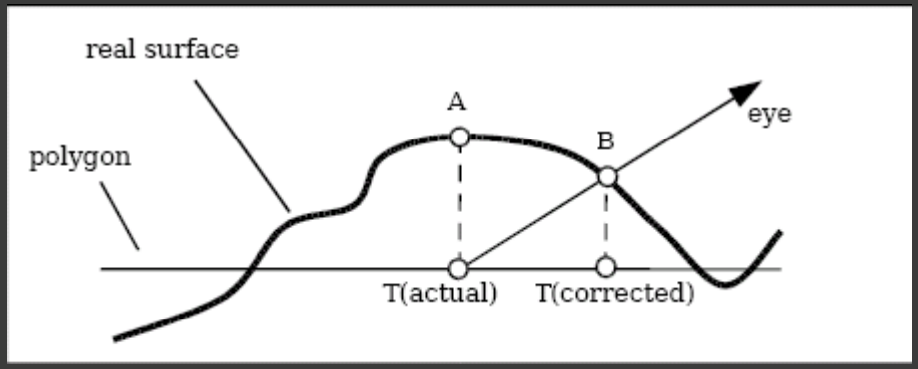


Normal mape se računaju tako da se uzme odgovarajući podatak o visini za tu poziciju(h(i, j)) u height mapi, te njene dvje susjedne točke(h(i + 1, j), h(i, j +1)). Nakon toga se izračunaju vektori od h(i, j) do h(i +1 , j) i h(i, j) do h(i, j + 1). Njihovim cross produktom se dobije normala koja će se spremit na poziciju i,j u normal mapu. Taj vektor se sprema tako što se njegove x, y i z komponente spremu respektivno u RGB komponentu teksture.Normala je definirana u intevalu od [-1,1]. Normala se sprema tako da se njenim x, y i z komponentama doda 1.0 te se potom pomnože s 0.5.

Parallax Mapping

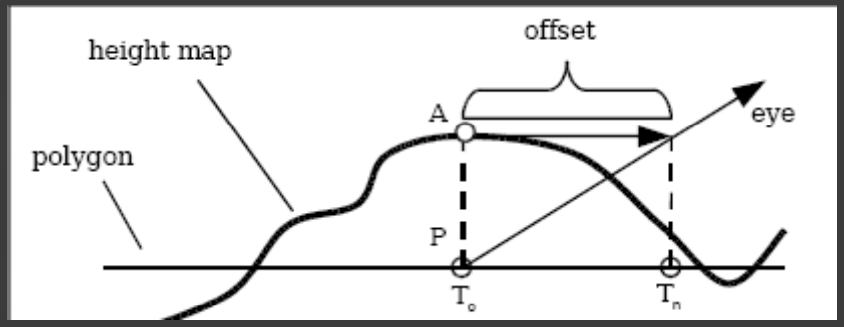
Parallax Mapping

Bump Mapping



Kada priljepimo teksturu koja bi trebala predstavljati neravnu površinu na poligon, vektorom pogleda vidimo točku A. No da je To bila stvarna geometrija mi bi vidjeli točku B. Kada bi koordinate teksture točke A mogle biti prepravljene mi bi mogli vidjeti točku B.

Odmicanjem svake koordinate posebno, možemo djelove s većom visinom odmaknuti od oka, a djelove s nižom visinom odmaknuti više prema oku.



Tri komponente su potrebne da bi se izračunao potreban odmak. Potreban je početna koordinata teksture, pripadna Vrjednost visine te vektor pogleda koji se nalazi u tekstur prostoru. Height mapa se koristi za prikaz visina površine, one se nalaze u intervalu [0, 1].

Nova koordinata teksture se računa po izaraz:

$$T_n = T_o + (h_{sb} \cdot V_{\{x,y\}} / V_{\{z\}})$$

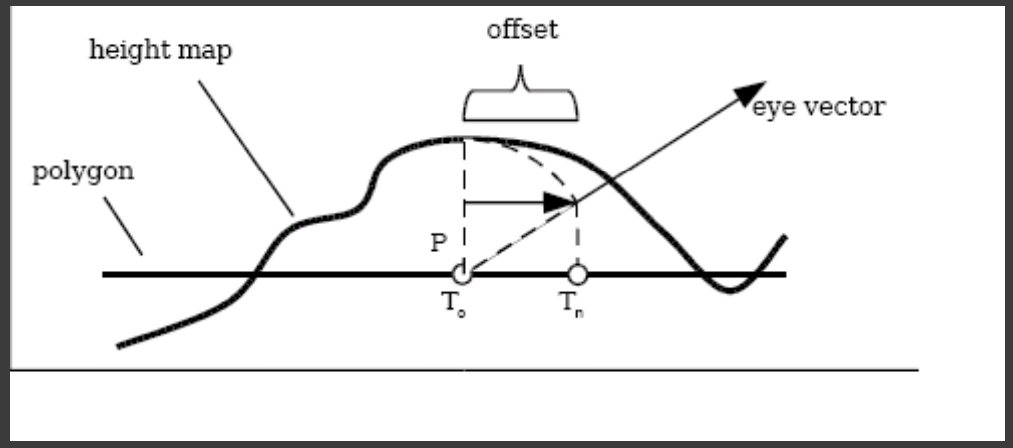
$$h_{sb} = h \cdot s + b$$

s predstavlja faktor skaliranja, a b bias koji se dodaje visini h.S i b se dodaju kako bi visina bila više prilagođena fizičkom izgledu površine koja se simulira.

Ovaj način računanja odmaka ima jako velii nedostatak što on predpostavlja da točka T0 i Tn imaju istu visinu, no ovo je rijetko kad slučaj.
Pri jakom malom kutu gledanja odmak postaje jako velik. Kada odmak postaje jako velik , razlika visina je najvjerojatnije jako velika što dovodi do preljevajnja pixela koji ne liče na orginalnu teksturu

Jednostavno rješenje ovome je da se ograniči odmak tako da ne može nikako narasti poviše hsb.To se postiže tako da se iz izraza za računanje nove koordinate teksture makne 1/V{z}

$$T_n = T_o + (h_{sb} \cdot V_{\{x,y\}})$$



Literatura:

1.Randi J. Rost, “**The OpenGL Shading Language**”, Second Edition, Addison-Wesley, December 2006

2.Dave Astle, “**More OpenGL Game Programming**”, Thomson Course Technology, 2006

3.Mark J. Kilgard, “**A Practical and Robust Bump-mapping Technique for Today’s GPUs**”, NVIDIA, July 2000

4.Terry Welsh, “**Parallax Mapping with Offset Limiting: A Per Pixel Approximation of Uneven Surfaces**”, Infiscape Corporation, January 2004