Realisztikus színtér

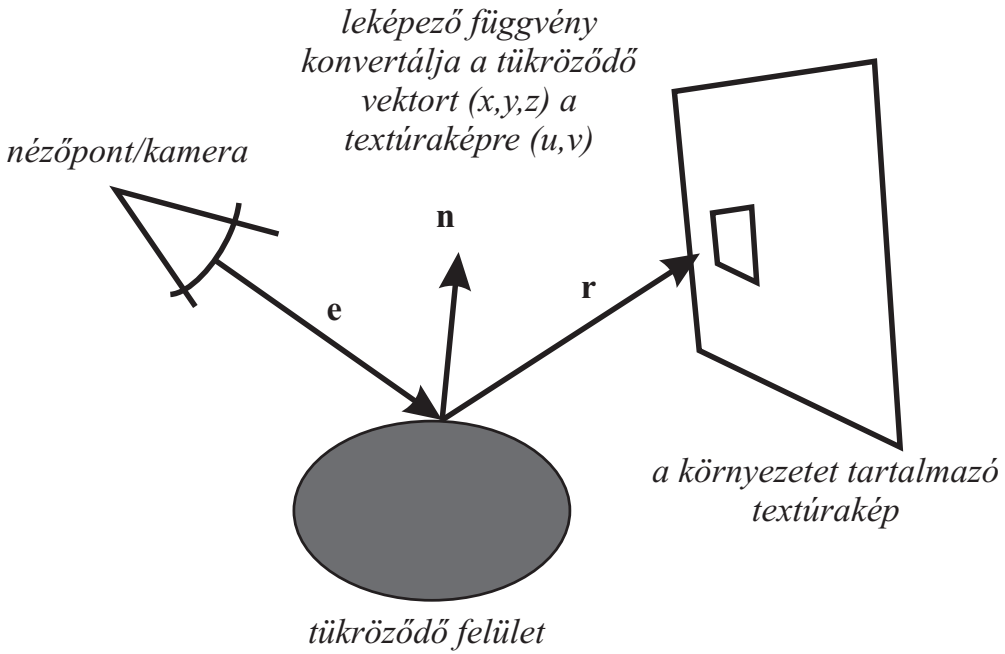
Környezet leképezés

Környezet leképezés

- Hatékony módszer görbe felületeken való tükröződés megjelenítésére
  - Egy sugarat indít a nézőpontból a tükröződő objektum egy pontjába
  - Ez a sugár ezután a pontban lévő normálvektor alapján visszaverődik
    - Ahelyett, hogy megkeresnénk a legközelebbi felülettel való metszését a visszavert fényvektornak az irányát használja a környezetet tartalmazó kép indexének a meghatározására

Környezet leképezés

- A kamera egy objektum felé néz
- Az **r** visszavert fényvektorát az **e** és **n** vektorokból számítjuk ki
- A visszavert fényvektor eléri a környezetet tartalmazó textúraképet
- Az elérési információt a leképező függvény felhasználásával számítjuk ki, amely az (x, y, z) visszatükröződő vektort alakítja át (u, v) értékre



Környezet leképezés

- A környezet leképezés feltételezi,
  - Az objektumok és fények, melyek a felületen tükröződnek, messze vannak
  - A tükröződő felület önmagát nem tükrözi
- A környezet leképezési algoritmus lépései a következőek
  1.) A környezetet ábrázoló kétdimenziós kép előállítása és betöltése
  2.) A tükröződő objektum mindegyik pixelére az objektum felületén lévő pozíciókban kiszámítjuk a normál egységvektorokat
  3.) A visszavert fényvektornak a kiszámítása a nézőpontvektor (nézőpont iránya) és a normál egységvektorból
  4.) A visszavert fényvektor segítségével meghatározzuk a környezeti térkép egy indexét, ami a környezet színe az objektum adott pontjában
  5.) A környezeti térképből kinyert texel adatokat használjuk fel az aktuális pixel színezésére

- Mindegyik leképezett pixelre kiszámítjuk a visszavert fényvektort és (ρ, ϕ) gömbi koordinátákba transzformáljuk azokat
  - A $\phi \in [0, 2\pi]$-t hosszúsági körnek nevezzük
  - $\rho \in [0, \pi]$-t szélességi körnek nevezzük
- (ρ, ϕ)-t a következő összefüggések alapján számítjuk ki

$$\rho = \arccos(-r_z)$$

$$\phi = \arctan(\frac{r_y}{r_x}), \text{ ha } r_x \neq 0$$

  - $\mathbf{r} = (r_x, r_y, r_z)$ a normalizált visszavert fényvektor

- A nézőponthoz tartozó visszavert fényvektort, hasonlóan számítjuk a fény tükröződési vektoréhoz

$$\mathbf{r} = \mathbf{e} - 2(\mathbf{n} \cdot \mathbf{e})\mathbf{n}$$

  - **e** a normalizált vektor a felület pozícióban
  - **n** az egység normálvektor az adott pozícióban

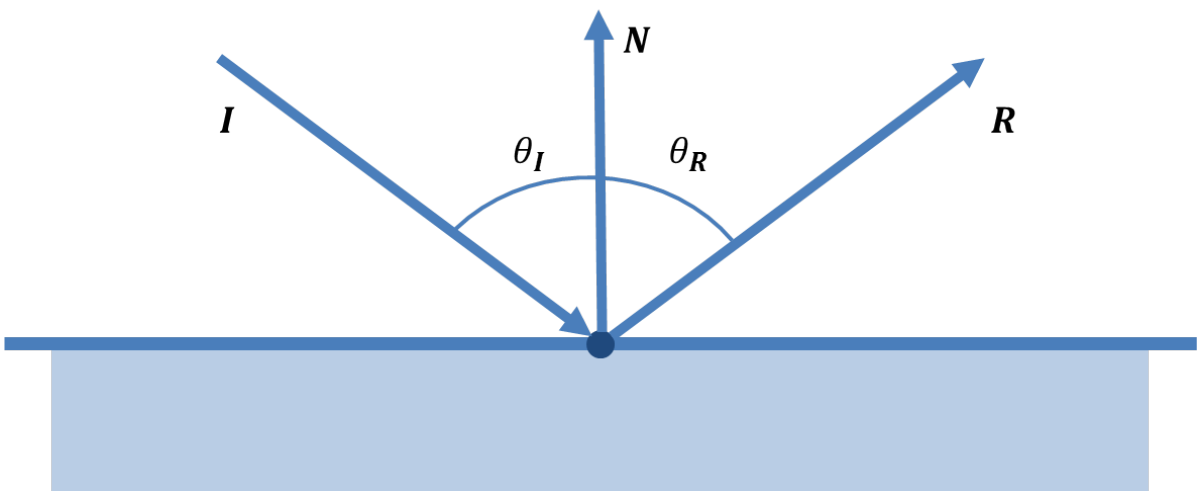
- A (ρ, ϕ) gömbi koordinátákat a $[0, 1)$ tartományra képezzük le
- (u, v) koordinátaként használjuk a környezet textúra eléréséhez, a tükröződő szín előállítására
- A tükröződési vektort transzformáljuk gömbi koordinátákba
  - A környezetet tartalmazó textúrakép egy „kiterített" gömb képe
- A textúra befed egy gömböt, ami körbeveszi a tükröződési pontot
- Ezt a leképező függvényt néha szélességi-hosszúsági leképezésnek is hívják
  - v a szélességi körökkel, u pedig a hosszúsági körökkel egyezik meg

- Hátrányok
  - $\phi = 0$-ban van egy határ
  - A térkép összefut a sarkoknál
- A környezet leképezésben használt képnek egyeznie kell a szegélyeknél a függőleges élek mentén
- El kell kerülni a torzítási problémákat a felső és alsó élek környezetében
- Használhatjuk az indexek kiszámítására a vertexekben és ezután interpolálhatjuk ezeket a koordinátákat
- Hiba fordul elő abban az esetben is, amikor egy háromszög vertexei olyan indexekkel rendelkeznek a környezeti térképen, melyek a sarkokon mennek keresztül

- A kamerát egy kocka középpontjában helyezzük el és levetítjük a környezetet a kocka oldalaira
- A környezeti térképet bármely renderelővel könnyen elő lehet állítani valós-időben
- A visszavert fényvektornak az iránya meghatározza, hogy a kocka melyik oldalát használjuk
  - A visszavert fényvektor abszolút értékben legnagyobb komponense meghatározza, hogy milyen kapcsolatban van az oldallal
    - Például a $(-3.2, 5.1, -8.4)$ a $-Z$ oldalt jelöli ki
  - A maradék két komponenst a legnagyobb komponens abszolút értékével elosztva, majd a $[0, 1]$ intervallumra leképezve kapjuk meg a textúra-koordinátákat a kiválasztott lapon

- Tükrözött vektorok kiszámítása
  - Beeső sugár I
  - Tükrözött sugár R
  - Felületi normál N
  - Tökéletes tükröződésnél $\theta_I = \theta_R$
  - $R = I - 2N(N \cdot I)$

```
float3 reflect (float3 I, float3 N)
{
  return I - 2.0 * N *dot(N, I);
}
```



- Konvex vagy majdnem konvex objektumok esetén működik jól
- Csak az iránytól függ nem pedig a pozíciótól
  - Sík tükröződő felületek esetén nem jól viselkedik
  - Legjobban a görbefelületeken alkalmazható

Cg - Vertex program

```
void C7E1v_reflection(float4 position : POSITION,
                      float2 texCoord : TEXCOORD0,
                      float3 normal   : NORMAL,
                  out float4 oPosition  : POSITION,
                  out float2 oTexCoord  : TEXCOORD0,
                  out float3 R          : TEXCOORD1,
                uniform float3   eyePositionW,
                uniform float4x4 modelViewProj,
                uniform float4x4 modelToWorld)
{
  oPosition = mul(modelViewProj, position);
  oTexCoord = texCoord;
  // A világtérben való pozíció és normálvektor kiszámítása
  float3 positionW = mul(modelToWorld, position).xyz;
  float3 N = mul((float3x3)modelToWorld, normal);
  N = normalize(N);
  // A beeső és tükröződő vektorok kiszámítása
  float3 I = positionW - eyePositionW;
  R = reflect(I, N);
}
```

Cg - Fragmens program

```
void C7E2f_reflection(float2 texCoord : TEXCOORD0,
                      float3 R        : TEXCOORD1,

                  out float4 color : COLOR,

                uniform float reflectivity,
                uniform sampler2D decalMap,
                uniform samplerCUBE environmentMap)
{
  // A tükröződő környezeti szín meghatározása
  float4 reflectedColor = texCUBE(environmentMap, R);

  // A matrica textúra szín meghatározása
  float4 decalColor = tex2D(decalMap, texCoord);

  color = lerp(decalColor, reflectedColor, reflectivity);
}
```

- A tükröződő vektor kiszámításához használhatnánk fragmens programot
  - Jobb képminőség érhető el
  - Spelkuláris megvilágításnál a tükröződési vektor nem-lineáris módon változik fragmensről-fragmensre
  - A lineárisan interpolált vertexenkénti tükröződési értékek nem megfelelőek
  - Az objektum körvonalánál artifaktumok jelennek meg
- Nem biztos, hogy észrevehető a különbség

- A textúraképet egy tökéletesen tükröződő gömbön megjelenő környezet ortogonális nézetéből állítjuk elő
  - A textúrát gömbtérképnek nevezzük
- Előállítása
  - Egy csillogó gömbről készítünk fényképet
    - Ezt az kör alakú eredmény gömbtérképet néha fényvizsgálatnak is hívják
  - A gömbtérképet szintetikus színtér esetén való előállítása
    - Sugárkövetéssel
    - A cube map környezeti térképnél használt képek gömbre való vetítésével

- A gömbtérképnek van egy bázisa
- A képet egy **f** tengely mentén nézzük a világtérben **u** felfele mutató vektorral és feltesszük, hogy a **h** vektor vízszintesen jobbra mutat (mindegyik vektor normalizált)
- Ez egy bázis mátrixot ad

$$\begin{pmatrix} h_x & h_y & h_z & 0 \\ u_x & u_y & u_z & 0 \\ f_x & f_y & f_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- A gömbtérkép egy elemének eléréséhez
  - Az $\mathbf{n}$ felületi normált és a szem pozíciójából a vertexbe menő $\mathbf{e}$ vektort transzformáljuk
  - Ez az $\mathbf{n}'$ és $\mathbf{e}'$ vektorokat állítja elő a gömbtérkép terében
- A visszavert fényvektort a következőképpen állítjuk elő

$$\mathbf{r} = \mathbf{e}' - 2(\mathbf{n}' \cdot \mathbf{e}')\mathbf{n}'$$

  - Ahol az $\mathbf{r}$ eredmény vektor a gömbtérkép terében van

- A tükröződő gömb a teljes környezetet mutatja meg, ami a gömb előtt található
  - Ez mindegyik visszavert irányt leképezi a gömb kétdimenziós képének egy pontjára
- Ha meg akarjuk határozni a tükröződési irányt a gömbtérkép egy adott pontjában
  - Szükségünk van a gömb pontjában a felületi normálvektorra

- Fordítsuk meg az eljárást és vegyük a gömbön a pozíciót
  - Vezessük le a felületi normált a gömbön, ami az (u, v) paramétereket határozza meg a textúra adatok eléréséhez
- A gömb normálvektora (r_x, r_y, r_z) a visszavert fényvektor és a szem iránya $(0, 0, 1)$ között fél úton található
  - Az $\mathbf{n}$ normálvektor egyszerűen felírható a szem és a visszavert fényvektor összegeként, amelyet ezután normalizálunk

$$m = \sqrt{r_x^2 + r_y^2 + (r_z + 1)^2},$$

$$\mathbf{n} = \left(\frac{r_x}{m}, \frac{r_y}{m}, \frac{r_z + 1}{m} \right)$$

- Hátrány
  - A gömbtérképen két pont közötti mozgás nem lineáris
  - A gömbtérkép csak egyetlen nézőpont irány esetén érvényes
    - Ha változik a nézőpont iránya, akkor a leképezést újra végre kell hajtani
  - Mivel a gömbtérkép nem tartalmazza a teljes környezetet
    - Előfordulhat, hogy képkockáról-képkockára ki kell számolni a környezeti leképezés textrúra-koordinátáit az új nézőpont irányra az alkalmazás szakaszban
  - Amennyiben a nézőpont iránya változik, akkor érdemesebb nézőpont független környezeti leképezést használni

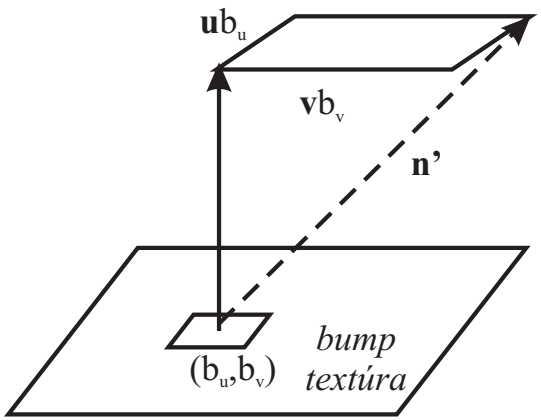
Felületi egyenetlenség leképezés - Bump mapping

Felületi egyenetlenség leképezés - Bump mapping

- A felületek megjelenését teszi egyenetlenné
  - Olyan tulajdonságot szimulálhat, amit ellenkező esetben sok poligon felhasználásával lehet csak modellezni
- Az alap ötlet az, hogy a textúra nem a szín komponenst változtatja meg a megvilágítási egyenletben, hanem a felületi normálvektorokat módosítja
  - A normálvektorokat egy textúrában tárolunk el
- A felületi geometria normálja változatlan marad
  - A megvilágítási egyenletben használt normálvektorokat változtatjuk meg pixelenként

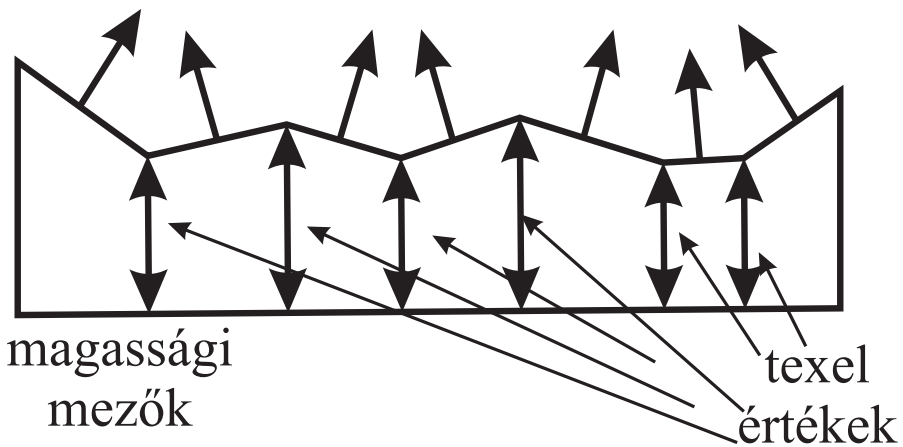
Felületi egyenetlenség leképezés - Bump mapping

- Az egyik felületi egyenetlenség textúrázási technika esetén b_u és b_v előjeles értékeket tárolnak el egy textúrában
- Ez a két érték a normál változásának a mennyiségét tárolja u és v tengelyek mentén
- Ezeket a textúra értékeket használjuk a normálisra merőleges két vektor skálázására
  - A textúra értékek általában bilineárisan interpoláltak
- A két b_u és b_v adja meg, hogy a felület milyen irányba néz a pontban



Felületi egyenetlenség leképezés - Bump mapping

- Magassági mezőket használunk a felületi normálvektorok irányainak a módosítására
- A szomszédos oszlopok különbsége adja meg az u, valamint a szomszédos sorok különbsége a v meredekségét



Felületi egyenetlenség leképezés - Bump mapping

- Meggyőző és olcsó módja a geometriai részletesség látszatának növelésére
- Hátrány
  - Az objektumok körvonalai körül azonban a hatás eltűnik
  - A felületi egyenetlenség alkalmazásakor az egyenetlenségek nem vetnek árnyékot a saját felületükön

Felületi egyenetlenség leképezés - Bump mapping

- Léteznek fejlettebb valósidejű renderelő módszerek, melyek használatával önárnyalási hatást is el lehet érni
  - Statikus színterek esetén a megvilágítást előre is ki lehet/kell számítani
  - Ha egy felületen nincs spekuláris megvilágítás és a fények nem mozognak a felülethez viszonyítva
    - A felületi egyenetlenséghez tartozó árnyalást ki lehet számítani egyszer és az eredményt egy szín textúraként használjuk az adott felületen
  - Ha a felület, fény és kamera mindegyike rögzítve vannak egymáshoz
    - A fényes, egyenetlen felületet elegendő egyszer előállítani

Cg - Vertex program

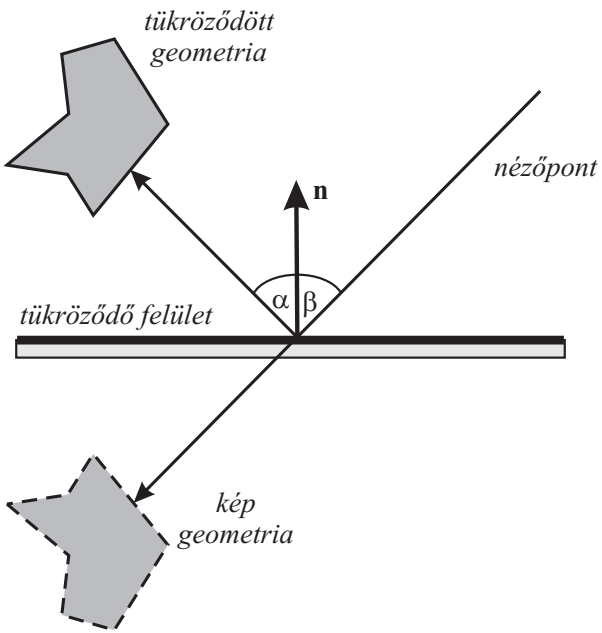
```
void C8E1v_bumpWall(float4 position : POSITION,
                    float2 texCoord : TEXCOORD0,

                  out float4 oPosition      : POSITION,
                  out float2 oTexCoord      : TEXCOORD0,
                  out float3 lightDirection : TEXCOORD1,

              uniform float3   lightPosition,  // Objektumtér
              uniform float4x4 modelViewProj)
{
  oPosition = mul(modelViewProj, position);
  oTexCoord = texCoord;
  // Különbség vektorok az objektumtér
  // megvilágítási irányhoz
  lightDirection = lightPosition - position.xyz;
}
```

Cg - Fragmens program

```
float3 expand(float3 v)
{
  return (v-0.5)*2;  // Kiterjesztése a vektornak
}

void C8E2f_bumpSurf(float2 normalMapTexCoord : TEXCOORD0,
                    float3 lightDir          : TEXCOORD1,

                out float4 color : COLOR,

            uniform sampler2D   normalMap,
            uniform samplerCUBE normalizeCube)
{
  // Normalizes light vector with normalization cube map
  float3 lightTex = texCUBE(normalizeCube, lightDir).xyz;
  float3 light = expand(lightTex);

  // A normál map textúra mintavételezése és kiterjesztése
  float3 normalTex = tex2D(normalMap, normalMapTexCoord).xyz;
  float3 normal = expand(normalTex);

  // Diffúz megvilágítás
  color = dot(normal,light);
}
```

Tükröződések

- Sík tükröződést könnyebb megvalósítani és végrehajtani, mint egy általános tükröződést
- Ideális tükröződő felületre érvényes a *tükröződési törvény*, amely szerint a beesési szög megegyezik a visszavert fény kilépési szögével
- Ennek a törvénynek köszönhetően az objektum tükrözött képe egyszerűen maga a tükrözött objektum



- A visszatükrözött sugár követése helyett a beeső fényt követhetjük a tükröződő felületen
  - Egy tükröződést előállíthatunk egy objektum másolatának a transzformálásával a tükröződő pozícióba
  - A pozíció és az irány figyelembevételével a fényforrásokat is tükrözni kell
- Ha feltesszük, hogy a tükröződő felület **n** normálvektora $(0, 1, 0)$ és ez az origón megy keresztül
  - A mátrix, ami erre a síkra tükröz egy egyszerű tükröző $\mathbf{S}(1, -1, 1)$ skálázó mátrix

- Általános esetben az **M** tükröződési mátrix
  - **n** normálvektor
  - A tükröződő felület **p** pontja

$$\mathbf{F} = \mathbf{R}(\mathbf{n}, (0,1,0))\mathbf{T}(-\mathbf{p})$$

  - A síkot eltoljuk a $\mathbf{T}(-\mathbf{p})$ transzformációval úgy, hogy az origón keresztül menjen
  - A tükröződő felület **n** normálvektorát forgatjuk, hogy párhuzamos legyen az $(0,1,0)$ y-tengellyel
    - A forgatást az $\mathbf{R}(\mathbf{n}, (0,1,0)$ felhasználásával hajtjuk végre
  - A tükröződő felület ezek után az $y = 0$ síkhoz lesz igazítva

$$\mathbf{M} = \mathbf{F}^{-1}\mathbf{S}(1,-1,1)\mathbf{F}$$

- A mátrixot újra kell számolni, ha a pozíció vagy a tükröződő felület irányítottsága megváltozik

- Először a megjelenítendő színtér **M**-mel transzformált tükröződő objektumait
- A színtér többi részét rajzoljuk ki a tükröződő felülettel együtt
- A tükröződő felületnek részlegesen átlátszónak kell lenni
  - Azért, hogy a tükröződés látható legyen
- Amennyiben éles szögben nézünk rá az adott színtérre, akkor a tükröződő geometria láthatóvá válhat
  - A „kilógó ” rész eldobásával ez a probléma megoldható
  - A legalkalmasabb ennek a problémának a megoldására a *stencil puffer* használata

OpenGl példa

```
// A szín és mélység puffer frissítésének letiltása
glDisable(GL_DEPTH_TEST);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);

// A stencil műveletek beállítása
glEnable(GL_STENCIL_TEST);
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
glStencilFunc(GL_ALWAYS, 1, 0xffffffff);

// Puffer 1-esekkel való feltöltése ott ahol a padló van.
drawFloor();

// Újra engedélyezése a szín és mélység puffernek.
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glEnable(GL_DEPTH_TEST);
```

OpenGl példa

```
// Csak oda rajzolunk, ahol 1-esek vannak
glStencilFunc(GL_EQUAL, 1, 0xffffffff);

// Az 1-esek maradnak a pufferben.
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

// A tükrözött dinó kirajzolása, ahol 1-esek vannak.
glPushMatrix();
glScalef(1.0, -1.0, 1.0);
setLightSourcePositions();
drawNinja();
glPopMatrix();
glDisable(GL_STENCIL_TEST);
```

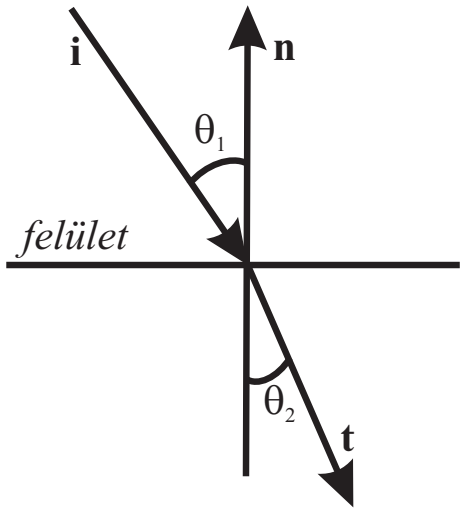
OpenGl példa

```
// Összemosott padló és az aktuális objektum kirajzolása

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glColor4f(0.7, 0.0, 0.0, 0.40);
drawFloor();
glDisable(GL_BLEND);
drawNinja();
```

- Egy másik probléma a hátsólap-eldobás miatt fordul elő
- Amennyiben a hátsólap-eldobás be van kapcsolva és egy objektumot skálázunk a tükröződési mátrixszal
  - A hátsólap-eldobás helyett az előlapok lesznek eldobva
- A megoldás az, hogy a hátsólap-eldobásból az előlap-eldobásra váltunk

- Snell törvénye
  - A beérkező és kilépő vektorok kapcsolatát állapítja meg
  - Az egyik közegből (mint például levegő) a másikba (például a víz) lép a fény

$$n_1 \sin(\theta_1) = n_2 \sin(\theta_2)$$



- n_k az adott közeg törésmutatója
- θ_k ($k \in \{1, 2\}$) a felületi normálishoz viszonyított szög

$$\mathbf{t} = r\mathbf{i} + (w - k)\mathbf{n}$$
$$r = n_1/n_2,$$
$$w = -(\mathbf{i} \cdot \mathbf{n})r,$$
$$k = \sqrt{1 + (w - r)(w + r)}$$

- A kiértékelés eléggé költséges
- A fénytörés mértéke a horizont környékén csökken
  - Kis bejövő szögek esetén a következő közelítést használhatjuk

$$\mathbf{t} = -c\mathbf{n} + \mathbf{i}$$

  - c víz szimulálása esetén 1.0 körül van
  - Ebben az esetben $\mathbf{t}$ vektort normalizálni kell

Cg példa - Vertex program

```
float3 refract (float3 I, float3 N, float etaRatio)
{
    float cosI = dot(-I, N);
    float cosT2 = 1.0f - etaRatio * etaRatio *
                (1.0f - cosI * cosI);
    float3 T = etaRatio * I +
            ((etaRatio * cosI - sqrt(abs(cosT2))) * N);
      return T * (float3)(cosT2 > 0);
}
```

Cg példa - Vertex program

```
void C7E3v_refraction(float4 position : POSITION,
                      float2 texCoord : TEXCOORD0,
                      float3 normal   : NORMAL,
                  out float4 oPosition  : POSITION,
                  out float2 oTexCoord  : TEXCOORD0,
                  out float3 T          : TEXCOORD1,
              uniform float etaRatio,
              uniform float3 eyePositionW,
              uniform float4x4 modelViewProj,
              uniform float4x4 modelToWorld)
{
  oPosition = mul(modelViewProj, position);
  oTexCoord = texCoord;

  // A pozíció és a normál kiszámítása a világtérben
  float3 positionW = mul(modelToWorld, position).xyz;
  float3 N = mul((float3x3)modelToWorld, normal);
  N = normalize(N);

  // A beeső és a kilépő vektorok kiszámítása
  float3 I = normalize(positionW - eyePositionW);
  T = refract(I, N, etaRatio);
}
```

Cg példa - Fragmens program

```
void C7E4f_refraction(float2 texCoord : TEXCOORD0,
                      float3 T        : TEXCOORD1,

                  out float4 color : COLOR,

              uniform float        transmittance,
              uniform sampler2D    decalMap,
              uniform samplerCUBE environmentMap)
{
  // Textúra szín betöltése
  float4 decalColor = tex2D(decalMap, texCoord);

  // A fénytörés által meghatározott szín
  float4 refractedColor = texCUBE(environmentMap, T);

  //A végső szín kiszámítása
  color = lerp(decalColor, refractedColor, transmittance);
}
```

- A Fresnel egyenletek azt írják le,
  - Mennyi fény verődik vissza és mennyi fény „törik” meg
- Alacsony szög esetén nagy a tükröződés és nincs/alig van fénytörés
  - Nem lehet látni mi van a víz alatt
- Realisztikusabb lesz az előállított kép
- A Fresnel egyenletek bonyolultak
- Empirikus közelítés
  - $reflectionCoefficient = max(0, min(1, bias + scale \times (0 + I \cdot N)^{power}))$

- A fénytörés egyszerűsítve volt
  - Függ a felszín normál vektorától
  - Függ a beesési szögtől
  - Függ a fénytörési hányadostól
- Fénytörés mértéke függ még a bejövő fény hullámhosszától
  - Például a vörös fény jobban elhajlik, mint a kék
  - Szimulálhatjuk, hogy mi történik a komponensekkel
    - Az adott komponensek sugaraihoz tartozó környezeti térkép értékét kell kikeresni
    - A fénytörési hányadosokat külön adjuk meg a vörös, zöld és kék komponensekre

Cg példa - Vertex program

```
void C7E5v_dispersion(float4 position : POSITION,
                      float3 normal   : NORMAL,
                  out float4 oPosition         : POSITION,
                  out float  reflectionFactor  : COLOR,
                  out float3 R                 : TEXCOORD0,
                  out float3 TRed              : TEXCOORD1,
                  out float3 TGreen            : TEXCOORD2,
                  out float3 TBlue             : TEXCOORD3,
              uniform float fresnelBias,
              uniform float fresnelScale,
              uniform float fresnelPower,
              uniform float3 etaRatio,
              uniform float3 eyePositionW,
              uniform float4x4 modelViewProj,
              uniform float4x4 modelToWorld)  {
  oPosition = mul(modelViewProj, position);
  // A pozíció és a normál kiszámítása a világtérben
  float3 positionW = mul(modelToWorld, position).xyz;
  float3 N = mul((float3x3)modelToWorld, normal); N = normalize(N);
  // A beeső, a visszavert és a kilépő vektorok kiszámítása
  float3 I = positionW - eyePositionW;
  R = reflect(I, N);  I = normalize(I);
  TRed   = refract(I, N, etaRatio.x);
  TGreen = refract(I, N, etaRatio.y);
  TBlue  = refract(I, N, etaRatio.z);
  // A tükröződési faktor kiszámítása
  reflectionFactor = fresnelBias + fresnelScale * pow(1 + dot(I, N),
                     fresnelPower ); }
```

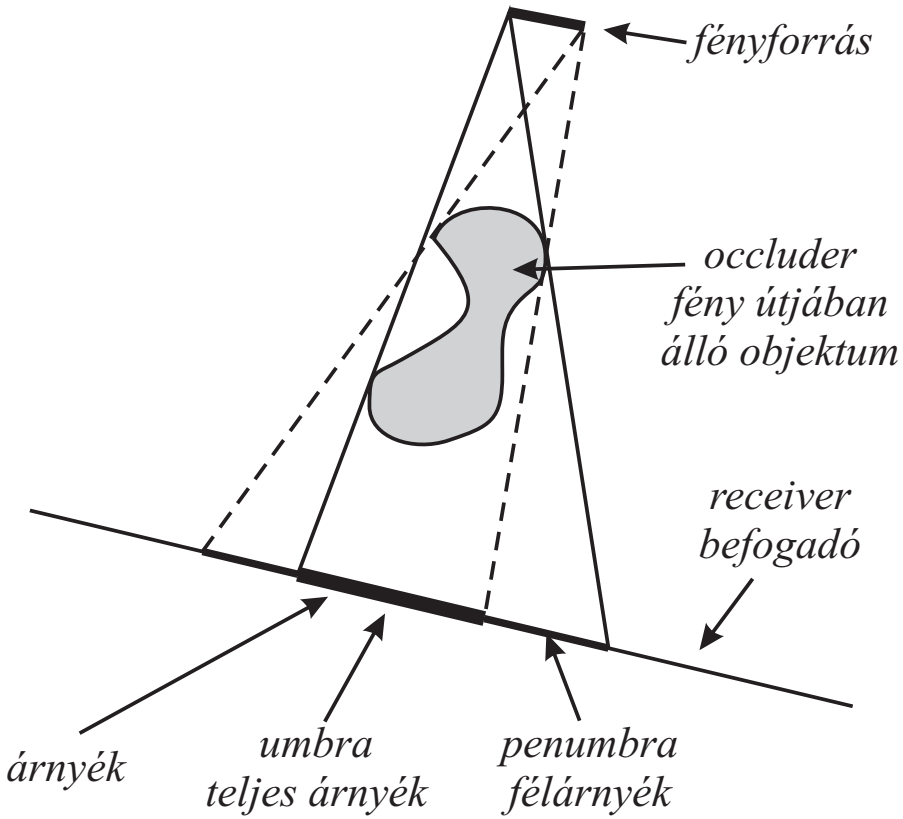
Cg példa - Fragmens program

```
void C7E6f_dispersion(float  reflectionFactor : COLOR,
                      float3 R                : TEXCOORD0,
                      float3 TRed             : TEXCOORD1,
                      float3 TGreen           : TEXCOORD2,
                      float3 TBlue            : TEXCOORD3,
                  out float4 color : COLOR,
              uniform samplerCUBE environmentMap0,
              uniform samplerCUBE environmentMap1,
              uniform samplerCUBE environmentMap2,
              uniform samplerCUBE environmentMap3)
{
  // A tükröződött szín betöltése
  float4 reflectedColor = texCUBE(environmentMap0, R);
  // A fénytörés színének kiszámítása
  float4 refractedColor;
  refractedColor.x = texCUBE(environmentMap1, TRed).x;
  refractedColor.y = texCUBE(environmentMap2, TGreen).y;
  refractedColor.z = texCUBE(environmentMap3, TBlue).z;
  refractedColor.w = 1;
  // A végső szín meghatározása
  color = lerp(refractedColor,
               reflectedColor,
               reflectionFactor);
}
```

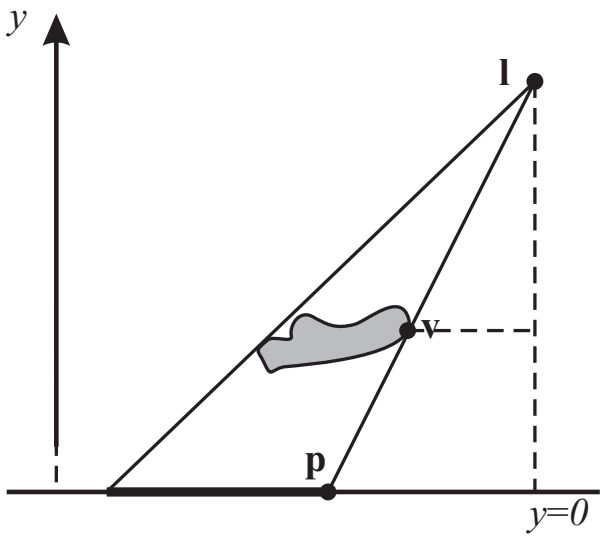
Árnyék síkfelületen

Árnyék síkfelületen

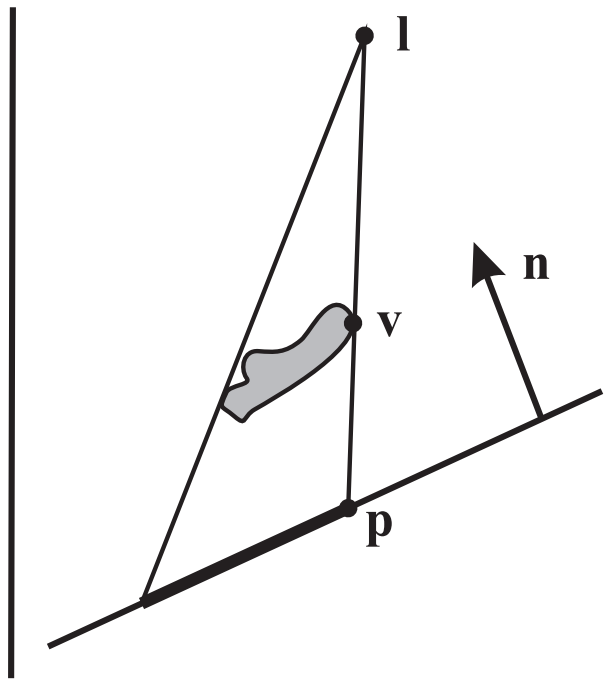
- Az árnyékok fontos elemei a valósághű képek előállításánál
  - A felhasználónak adnak némi információt az objektumok elhelyezéséről



- Egy egyszerű esete az árnyalásnak
  - Az objektumok árnyékai egy sík felületen jelennek meg
- Egy mátrixot hozunk létre
  - Az objektum vertexeit vetíti le egy síkra
  - Az árnyék előállításakor a háromdimenziós objektumot kétszer jelenítjük meg



$y = 0$ síkra vetett árnyék.



$\pi : \mathbf{n} \cdot \mathbf{x} + d = 0$ síkra vetett árnyék.

- Az x koordináták vetítésével kezdjük a levezetést
  - Hasonló háromszögek alapján a következő egyenlőségeket írhatjuk fel
    - $\frac{p_x - l_x}{v_x - l_x} = \frac{l_y}{l_y - v_y} \Longleftrightarrow p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$
- A z koordinátát hasonlóan kapjuk
  - $p_z = (l_y v_z - l_z v_y)/(l_y - v_y)$
- y koordináta 0-val egyenlő
- $\mathbf{M}$ projekciós mátrixot a következőképpen írhatjuk fel

$$\mathbf{M} = \begin{pmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{pmatrix}$$

- Általános esetben a sík egyenlete, amelyikre az árnyék vetődni fog
  - $\pi : \mathbf{n} \cdot \mathbf{x} + d = 0$
  - $\mathbf{v}$ pontot vetíti le $\mathbf{p}$ pontba

$$\mathbf{p} = \mathbf{l} - \frac{d + \mathbf{n} \cdot \mathbf{l}}{\mathbf{n} \cdot (\mathbf{v} - \mathbf{l})}(\mathbf{v} - \mathbf{l})$$

- Mátrix alakba felírva

$$\mathbf{M} = \begin{pmatrix} \mathbf{n} \cdot \mathbf{l} + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \mathbf{n} \cdot \mathbf{l} + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \mathbf{n} \cdot \mathbf{l} + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \mathbf{n} \cdot \mathbf{l} \end{pmatrix}$$

- Az általános mátrixba behelyettesítve
  - Az $y = 0$ síkhoz tartozó speciális értékeket
    - $\mathbf{n} = (0, 1, 0)^T$
    - $d = 0$
  - A speciális $\mathbf{M}$ mátrixot kapjuk
- Az árnyék előállításához egyszerűen ezt a mátrixot kell alkalmaznunk az objektumokra
  - A π síkra vetnek árnyékot
- Sötét színnel és megvilágítás nélkül kell megjeleníteni a vetületeket
  - A fény útjában álló objektumot kétszer rendereljük
    - Először a vetített poligonokat árnyékként
    - Másodszor pedig az eredeti objektumként

- Szükség van egy olyan módszerre, amely megakadályozza azt, hogy a vetített poligonokat a befogadó felület mögött állítsuk elő
  - A talajt rajzoljuk ki először
  - Aztán a vetített poligonokat kikapcsolt Z-puffer ellenőrzéssel
  - Azután az összes többi geometriát
- A vetített árnyék a síkon kívül is megjelenhet
  - A megoldása is hasonló az ott alkalmazott módszerrel, el kell távolítani a kilógó részt

OpenGL példa

```
void RenderScene(void)
  {
    // A szín puffer és a mélység puffer törlése
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // A sík felület (talaj) kirajzolása
    DrawGround();

    glPushMatrix();

    // A jet kirajzolása az új pozícióban
    // a fényforrás megfelelő pozícióba helyezése
    glEnable(GL_LIGHTING);
    glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    DrawJet(FALSE);

    glPopMatrix();
```

OpenGL példa

```
void RenderScene(void)
    {
...

    // Az árnyék rajzolása a talajon
    // A mélység ellenőrzés és a fény számítások tiltása
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);
    glPushMatrix();

    // Az árnyék vetítési mátrixszal való szorzás
    glMultMatrixf((GLfloat *)shadowMat);

    // Az árnyék beforgatása
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    // Árnyék rajzolása
    DrawJet(TRUE);

    glPopMatrix();
```

OpenGL példa

```
void RenderScene(void)
    {
...

    // A fényforrás kirajzolása
    glPushMatrix();
    glTranslatef(lightPos[0],lightPos[1], lightPos[2]);
    glColor3ub(255,255,0);
    glutSolidSphere(5.0f,10,10);
    glPopMatrix();

    // A mélység teszt engedélyezése
    glEnable(GL_DEPTH_TEST);

    // Az eredmény megjelenítése
    glutSwapBuffers();
    }
```

- Hátrány
  - Csak sík felületek esetén működik
  - Az árnyékot mindegyik képkocka esetén elő kell állítani
    - Még akkor is, ha az árnyék nem változik
- Egy jól működő módszer az, amikor az árnyékot egy textúrában állítjuk elő
  - Egy textúrázott téglalapként jelenítünk meg
  - Csak akkor kell újra kiszámítani, ha az árnyék megváltozik
    - A fényforrás vagy a fény útjában álló objektum vagy a befogadó felület mozog

Összefoglalás

- Környezeti leképezés
  - Blinn és Newell módszere
  - Cube map környezet leképezés
  - Sphere map környezet leképezés
- Felületi egyenetlenség
- Tükröződések
  - Sík tükröződés
  - Fénytörés
    - Snell törvénye
    - Fresnel hatás
    - Kromatikus szóródás
- Árnyék sík felületen
  - Vetített árnyék