Dr. Mileff Péter

GRAFIKA PROGRAMOZÁSA

FÉNYEK A SZÁMÍTÓGÉPES GRAFIKÁBAN II. RÉSZ

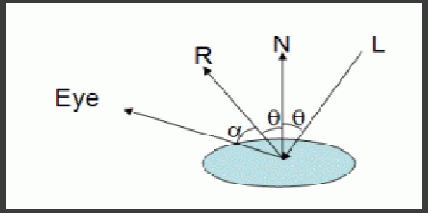
Miskolci Egyetem
Általános Informatikai Tanszék

PONTOSABB VERTEX SHADER ALAPÚ ÁRNYALÁS...

Pontosabb vertex shader alapú árnyalás

- **Cél:**
  - A korábbi modelltől komplexebb árnyalási modell
  - áttekintése és megvalósítása,
  - ahol már felhasználjuk a felület anyagtulajdonságait is.
  - Vertex shader alapú számítás
- A bemutatott példa a Phong modell egy egyszerűsített változatát alkalmazza
  - a **Blinn-Phong**-féle megközelítést

Phong-féle megközelítés



- a specular komponenst arányos a fény visszaverődési és a fény vektorok által bezárt szög koszinuszával
- **Jelölés:**
  - *L* jelenti azt a fény felől érkező vektort, amely a vertex-re esik.
  - *N* a vertek normálisa,
  - Eye vektor a vertex-ből a szembe mutat (kamera vektor),
  - *R* az *L* vektor visszavert komponense.
  - Az árnyalás specular intenzitása a koszinusz alfával egyezik meg.

Phong-féle megközelítés

- Ha az *Eye* vektor éppen egybeesik a visszaverődési vektorral,
  - akkor maximális specular intenzitást kapunk (cos(0)).
- Ahogyan az Eye vektor távolodik az *R* vektortól
  - úgy változik („bomlik”) az árnyalás specular komponense.
  - Ezt a változást egy úgynevezett *Shininess* faktor határozza meg.
    - „Fényerősség” faktora
  - Minél magasabb ez a faktor, annál hamarabb gyorsabb a változás (eltűnés).
- Az OpenGL ezt a shininess értéket 0 és 128 tartomány között értelmezi.

Phong-féle megközelítés

Shininess = 8 Shininess = 64 Shininess = 128

Phong-féle megközelítés

- *R* vektor kiszámítása:

$$R = -2N(L \cdot N) + L$$

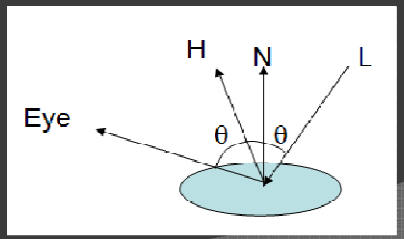
- A spekuláris komponens kiszámítása az OpenGL Phong modellje alapján:

$$Spec = (R \cdot Eye)^{s} * L_s * M_s$$

  - **Értelmezés:**
    - *s* kitevő jelenti a shininess értéket,
    - L_s a spekuláris fény intenzitása
    - M_s pedig az anyag spekuláris komponense

Blinn-Phong-féle megközelítés

- A Phong modell egyszerűsítése
- **Újítása:**
  - bevezetett egy gyorsabb és egyszerűbb algoritmust az árnyalás számítására
  - az úgynevezett „fél vektor” (half-vector) alkalmazásával.



Blinn-Phong-féle megközelítés

- **A spekuláris komponens intenzitása:**
  - a normálvektor(*N*) és a félvektor (*H*) által bezárt szög koszinuszaként számítjuk ki.
  - A *H* vektor formulája így lényegesen egyszerűbb, mint a valós Phong modellben lévő *R* vektoré:

$$H = Eye - L$$

- A spekuláris komponens ez alapján:

$$Spec = (N \cdot H)^s * L_s * M_s$$

A GLSL lehetőséget ad a félvektor kiszámítására is.

9

Példa shader program

- **Vertex árnyaló:**

```
void main() {
    vec3 normal, lightDir, viewVector, halfVector;
    vec4 diffuse, ambient, globalAmbient, specular = vec4(0.0);
    float NdotL,NdotHV;

    /* first transform the normal into eye space and normalize the result */
    normal = normalize(gl_NormalMatrix * gl_Normal);

    /* now normalize the light's direction. */
    lightDir = normalize(vec3(gl_LightSource[0].position));

    /* compute Lambert factor and clamp the result to the [0,1] range. */
    NdotL = max(dot(normal, lightDir), 0.0);
```

10

Példa shader program

- **Vertex árnyaló (folyt.):**

```
    /* Compute the diffuse, ambient and globalAmbient terms */
    diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
    ambient = gl_FrontMaterial.ambient * gl_LightSource[0].ambient;
    globalAmbient = gl_LightModel.ambient * gl_FrontMaterial.ambient;

    /* Compute the specular term if NdotL is larger than zero */
    if (NdotL > 0.0) {
        NdotHV = max(dot(normal, normalize(gl_LightSource[0].halfVector.xyz)),0.0);
        specular = gl_FrontMaterial.specular * gl_LightSource[0].specular *
                pow(NdotHV,gl_FrontMaterial.shininess);
    }
     /* Construct vertex front color */
    gl_FrontColor = globalAmbient + NdotL * diffuse + ambient + specular;
    gl_Position = ftransform();
}
```

11

Példa shader program

- **Fragment árnyaló:**

```
void main(){
    gl_FragColor = gl_Color;
}
```

12

Eredmény

PIXEL ALAPÚ ÁRNYALÁS…

Pixel alapú árnyalás
(fragmentumonkénti)

- **Jellemzője:**
  - nem a vertex-ekben számoljuk ki az intenzitás értéket,
  - csak az ahhoz szükséges vektorokat határozzuk meg.
  - Ezeket interpoláljuk a vertexek között elhelyezkedő pixelekre.
- Tehát az intenzitásokat így pixelenként számítjuk ki!
- **Eredménye:**
  - jelentős minőségi javulást eredményez a vertex alapú megoldásokhoz képest.
  - A megoldás jóval lassabb.

Pixel alapú árnyalás
(fragmentumonkénti)

- **Példa:** pontszerű fényforrás alkalmazása
- **Értelmezése:**
  - létezik egy pont, maga a fényforrás, ahonnan a sugarak minden irányba indulnak.
  - A sugarak intenzitása, a tárgyakon érzékelhető fény erőssége függ a fényforrás távolságától.
- A megvalósítás során tehát ezt a két különbséget kell figyelembe venni.
- Az első különbség orvoslása:
  - minden vertex-re kiszámoljuk a vertex és a fényforrás különbségének vektorát.
  - Tehát az objektumra nem fog párhuzamosan esni a fény.

Pixel alapú árnyalás
(fregmentumonkénti)

- **A fény távolságtól függő intenzitása:**

$$att = \frac{1}{k_0 + k_1 d + k_2 d^2}$$

light attenuation

  - k_0 a konstans, a k_1 a lineáris, a k_2 pedig a kvadratikus lecsengés faktora,
  - d a vertex fénytől való távolsága.
  - A lecsengés értékének számítása:
    - Nem tudjuk a vertex árnyaló segítségével kiszámítani és annak interpolált értékét felhasználni a fragment árnyalóban,
    - **Oka:** nem lineárisan változik a távolsággal.
    - De a vertex árnyalóban kiszámított távolság interpolált értékeit felhasználhatjuk a fragment árnyalóban a lecsengés kiszámítására

17

Pixel alapú árnyalás
(fregmentumonkénti)

- **Egy pixel színének kiszámítása tehát:**

$$color = ambientGlobal + att(ambient + diffuse + specular)$$

Globális megvilágítás — **Távolság függvényében változó tag**

18

Pixel alapú árnyalás
(vertex árnyaló)

```
varying vec4 diffuse,ambientGlobal,ambient;
varying vec3 normal,lightDir,halfVector;
varying float dist;

void main() {
    vec4 ecPos;
    vec3 aux;

    /* first transform the normal into eye space and normalize the result */
    normal = normalize(gl_NormalMatrix * gl_Normal);

    /* Compute the point type light's direction */
    ecPos = gl_ModelViewMatrix * gl_Vertex;
    aux = vec3(gl_LightSource[0].position-ecPos);
    lightDir = normalize(aux);
```

Pontszerű fényforrás értékeinek számítása Vertexen-ként

19

Pixel alapú árnyalás
(vertex árnyaló)

```
    /* Normalize the halfVector to pass it to the fragment shader */
    halfVector = normalize(gl_LightSource[0].halfVector.xyz);

    /* Compute the diffuse, ambient and globalAmbient terms */
    diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
    ambient = gl_FrontMaterial.ambient * gl_LightSource[0].ambient;
    ambientGlobal = gl_LightModel.ambient * gl_FrontMaterial.ambient;

    gl_Position = ftransform();
}
```

20

Pixel alapú árnyalás
(pixel árnyaló)

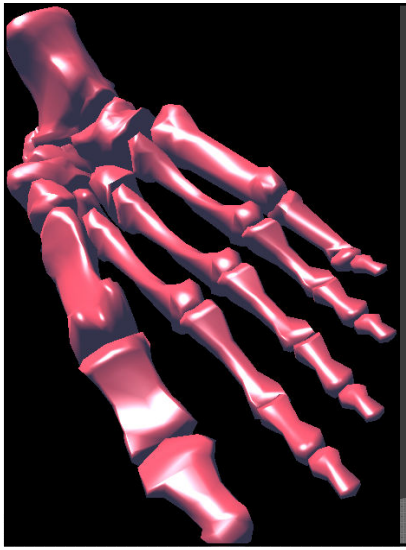
```
varying vec4 diffuse,ambientGlobal, ambient;
varying vec3 normal,lightDir,halfVector;
varying float dist;

void main() {
    vec3 n,halfV,viewV;
    float NdotL,NdotHV,att;
    vec4 color = ambientGlobal;

    n = normalize(normal);

   /* compute the dot product between normal and lightDir */
    NdotL = max(dot(n,normalize(lightDir)),0.0);
```

21

Pixel alapú árnyalás
(pixel árnyaló)

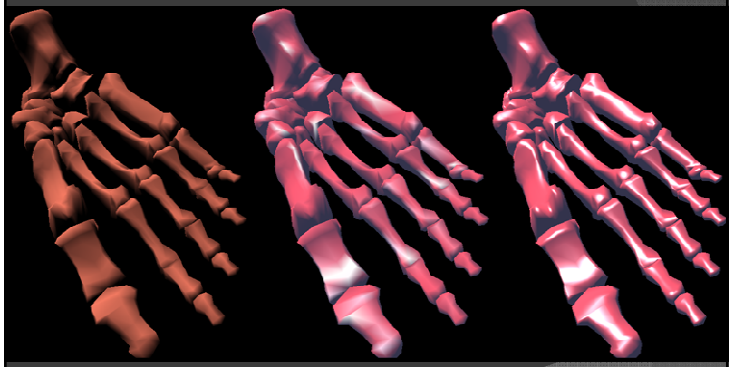
```
   /* Calculate attenuation */
    att = 1.0 / (gl_LightSource[0].constantAttenuation +
                 gl_LightSource[0].linearAttenuation * dist +
                 gl_LightSource[0].quadraticAttenuation * dist * dist);

     color += att * (diffuse * NdotL + ambient);

    halfV = normalize(halfVector);
    NdotHV = max(dot(n,halfV),0.0);
    color += att * gl_FrontMaterial.specular * gl_LightSource[0].specular *
                 pow(NdotHV,gl_FrontMaterial.shininess);

    gl_FragColor = color;
}
```

22



Eredmény

23



Egyszerű árnyalás | Vertex-enkénti árnyalás | Pixel alapú árnyalás

24

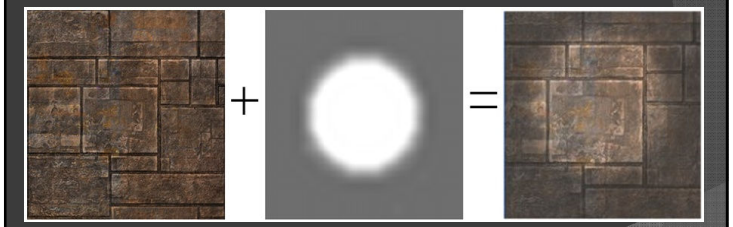
FÉNYTÉRKÉPEK…

Fénytérképek

- „Szegény ember” megvilágítási modellje
- A mai játékok grafikai világa komplex
  - a megvilágítás kiszámítása jelentős számításigénnyel rendelkezik.
  - Az egyszerre megjelenített poligonok száma elérheti az 1 milliót is,
  - melyeket a színtértől és a valósághűbb modellezéstől függően egyszerre több fényforrás is megvilágíthat.
- A korai számítógépek (kb. 1997-2006) erőforrásai jelentősen korlátozva voltak
  - Emiatt alternatív megvilágítási modellre volt szükség
    - a központi processzor és a GPU tehermentesítésére
  - **Lightmapping (fénytérképek)**

Fénytérképek

- **Fénytérkép:** valójában olyan textúrák, amelyek az adott „falrészletre eső fény intenzitását tárolják”.



Diffúz textúra | Fénytérkép | Mixelt textúra

Fénytérképek

- **Alkalmazásuk:**
  - előre kiszámítjuk (nem valós időben) a vertexek megvilágítottságát a fény és a vertex távolságából,
  - ezek összességét egy vagy több textúrában tároljuk el.
- **A folyamat egy mintavételezési eljárás**
  - „bejárjuk” a poligont és megnézzük, hogy melyik elemre milyen intenzitású fény esik.
  - Minél kisebb a bejárás egysége, annál részletesebb textúrát kapunk.
- **Megjelenítés menete:**
  - a textúrát, vagy textúrákat használjuk fel a felület kirajzolás során egy plusz textúraként
    - **Multitexturing**
  - így szimulálva a megvilágítást

Fénytérképek

- **A megoldás előnye:**
  - alacsony teljesítménnyel rendelkező számítógépeken is képes volt megfelelő sebességet nyújtani.
  - Az akkori hardverek több textúrát már képesek voltak gyorsan kezelni,
    - így kiváló megoldás jelentett a statikus fények modellezésére.
- **A megoldás hátránya:**
  - az előre generált fények miatt a megvilágítás statikus
  - Dinamikus fények (pl. mozgó lámpa) megvalósítására nem alkalmas.
- Az első számítógépes játék, amely ilyen technikát alkalmazott a **Quake** volt – 1996
  - még GPU nélkül
- Ettől fogva egyeduralkodóvá vált a játékszoftverekben.

29

Fénytérképek tárolása

- Több megoldás is kialakult.
- **A leginkább elterjedt:**
  - a virtuális világ teljes fénytérképét egy nagy (pl. 2048x2048) textúrában tárolják le
  - megjelenítés során az u,v koordináták alapján kerülnek a megadott részek a megadott falra.
- **E technika hátránya:** ha a bejárható világ mérete nagy, sok esetben egy textúra már nem elegendő.
  - Sok játékoknál azonban kis fénytérkép méret nem jelent problémát
    - kirajzolás közben a GPU elnyújtotta azt bilineáris (vagy trilineáris) szűrést is végrehajtva.
    - A végeredmény tehát elmosódik.
    - Az eredeti textúrával összetéve általában ez nem jelent gondot, hiszen a valós életben is sok árnyék széle elmosódott.

30

Fénytérképek a mai játékokban

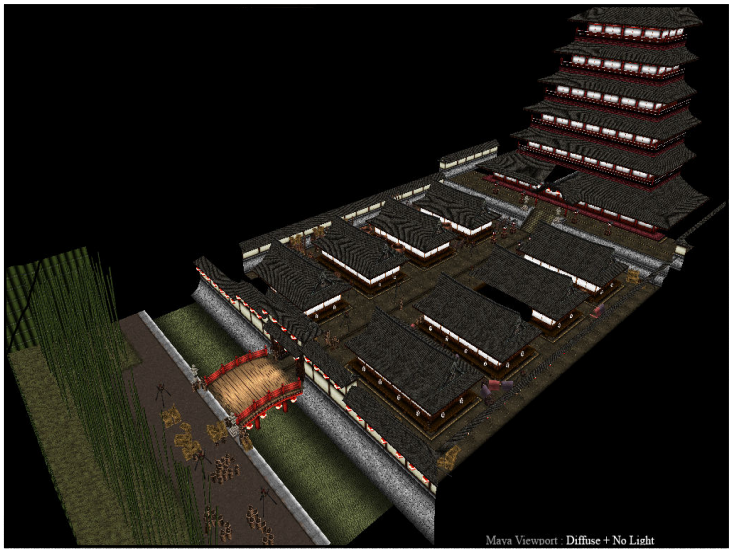
- A mai játékok komplexek – sok vertek, magas képi minőség
  - A fejlődés során látszik az eltolódás a valósidejűség felé
  - Egyre több dinamikus fényforrással dolgoznak a játékok
- **Cél az egyensúly megtalálása:**
  - a minőség és a teljesítmény között
  - Nem biztos, hogy célszerű csak dinamikus fényforrásokkal dolgozni.
  - A jó minőségű fénytérképek megjelenítése sokkal kevesebb teljesítményt igényel még ma is!

31

Fénytérképek a mai játékokban

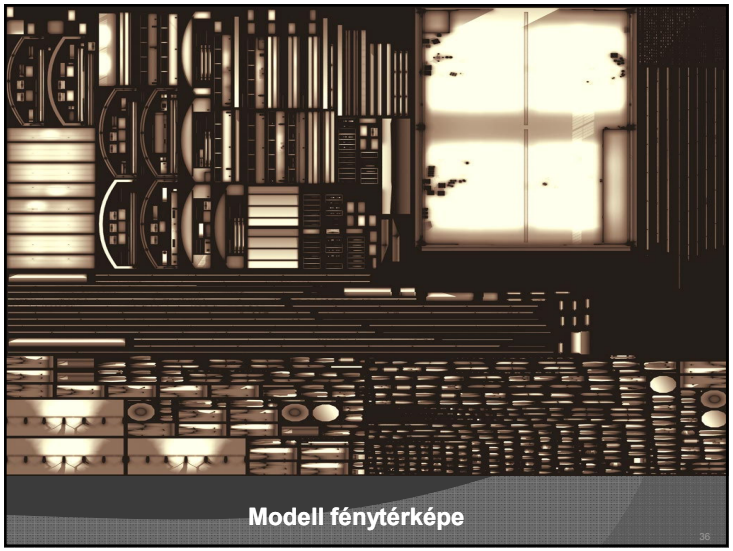
- **A fénytérképek szükségesek:**
  - Alátámasztja a tényt, hogy minden mai modern grafikus motorban és szerkesztő szoftverben megtalálható
    - pl. Unity, Unreal, CryEngine, Blender, 3D Studio)
  - Egyszerűbb/célszerűbb egy világot fénytérképekkel és néhány fénnyel megvalósítani
  - mint fények százaival dolgozni, azt optimalizálni, máshonnan erőforrást elvenni.
  - A mobil és tábla eszközök térnyerése:
    - az alacsonyabb számítási kapacitás miatt a technika újból fénykorát éli.

32


Maya Viewport : Diffuse + No Light

iShuriken
Maya Viewport : Diffuse + Lightmap + No Light

Fénytérképezett modell
35


Modell fénytérképe
36

GAME OVER