Modellezés

- OpenGL állapotmodell
  - Amikor egy állapot értéke be van állítva, az addig nem változik meg, amíg egy másik függvény meg nem változtatja azt
  - Két lehetséges értékű állapotok
    - `glEnable()`
    - `glDisable()`
  - Különböző értékek
    - Lekérdezés pl. `glGetFloatv(GLenum pname, GLfloat *params)`

Egy objektum modellezése

- Az objektumot felépítő primitívek vertexeit külön-külön is megadhatjuk
  - Több ezer primitívből álló alakzatnál már problémás
- Bonyolult alakzatokat és azok attribútumait egy jól meghatározott struktúrájú adatszerkezetben tároljuk
  - Egyetlen függvényhívás segítségével jelenítjük meg
- Szabályok
  - Poligonoknak síkbeli alakzatoknak kell lenniük
  - Poligon élei nem metszhetik egymást
  - Poligonnak konvexnek kell lennie

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

```
//...
// Négyszögsáv kezdete
glBegin(GL_QUAD_STRIP);
// Egy kör mentén számítjuk ki a henger
// palástjának a vertex pontjait
  for(angle = 0.0f; angle <= (2.0f*GL_PI);
      angle -= 2*(GL_PI/n)) {
// A következő vertex x és y pozíciójának
// a kiszámítása R = 1 esetén
    x = cos(angle);
    y = sin(angle);
//...
// A négyszögsáv alsó és felső pontjainak
// a megadása
    if (h1 < h2) {
      lVertex3f(R1*x, R1*y, h1);
      glVertex3f(R2*x, R2*y, h2);
    } else {
      glVertex3f(R2*x, R2*y, h2);
      glVertex3f(R1*x, R1*y, h1);
    }
  }
glEnd();
```

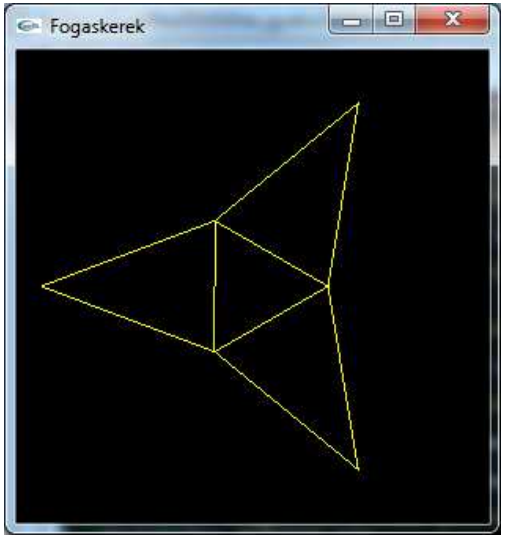
- Nézőponttól vett távolság eltárolása
  - z érték összehasonlítása azzal a z értékkel, amit már korábban eltároltunk
  - z értékek tárolását egy szín pufferrel megegyező méretű pufferrel
- Bizonyos esetekben szükséges a mélységpuffer írásának ideiglenes felfüggesztése
  - `glDepthMask(GL_FALSE)`
  - A mélység teszt ugyanúgy végrehajtódik a korábbi értékekkel

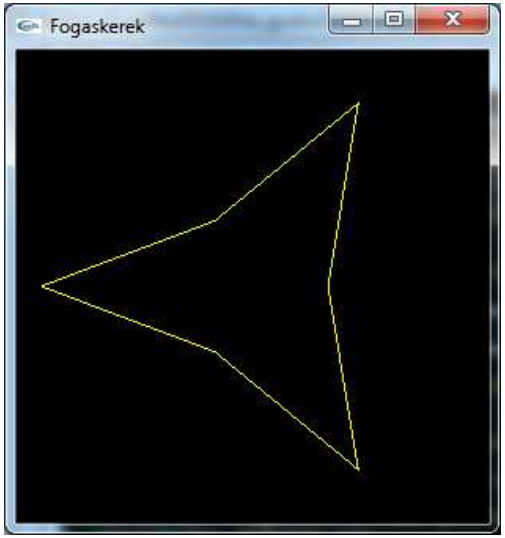
- Alapesetben a mélységellenőrzés a kisebb relációt használ a nem látható objektumhoz tartozó pixelek eltávolítására
- Lehetőség van az összehasonlító reláció megadására
  - `glDepthFunc(GLenum func)`

`GL_NEVER`	Mindig hamis
`GL_LESS`	Igaz, ha a bejövő mélység érték kisebb, mint az eltárolt érték
`GL_EQUAL`	Igaz, ha a bejövő mélység érték megegyezik az eltárolt értékkel
`GL_LEQUAL`	Igaz, ha a bejövő mélység érték kisebb vagy egyenlő, mint az eltárolt érték
`GL_GREATER`	Igaz, ha a bejövő mélység érték nagyobb, mint az eltárolt érték
`GL_NOTEQUAL`	Igaz, ha a bejövő mélység érték nem egyenlő az eltárolt értékkel
`GL_GEQUAL`	Igaz, ha a bejövő mélység érték nagyobb vagy egyenlő, mint az eltárolt érték
`GL_ALWAYS`	Mindig igaz

- A `GL_EQUAL` és `GL_NOTEQUAL` relációk esetén meg kell változtatni az alap $[0.0 - 1.0]$ mélység értékek tartományát
  - `glDepthRange(GLclampd nearVal, GLclampd farVal)`
    - Első paramétere a közeli vágósík
    - Második paramétere a távoli vágósík
  - A vágás és a homogén koordináták negyedik w elemével való osztás után, a mélység értékek a $[-1.0, 1.0]$ tartományba képződnek le
    - A `glDepthRange` adja meg a lineáris leképezését ezeknek a normalizált mélység koordinátáknak az ablak mélységértékeire nézve

- Drótvázas megjelenítéskor a belső alakzat éleit nem kell megjeleníteni
  - Az él flag-et hamisra kell állítani
- `glEdgeFlag`
  - `TRUE`
  - `FALSE`





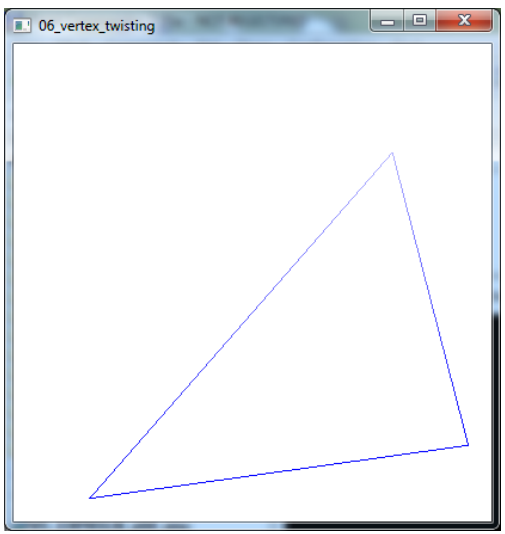
```
glBegin(GL_TRIANGLES);
for(i=0, angle=0.0; i<n; i++, angle += D_Angle)
{
   glEdgeFlag(FALSE);
   glVertex2f(x+radius* cos(angle),
              y+radius*sin(angle));
    glEdgeFlag(TRUE);
   glVertex2f(x+radius*cos(angle+Delta_Angle),
              y+radius*sin(angle+Delta_Angle));
   glVertex2f(x+2.8*radius*cos(angle+Half_DAngle),
              y+2.8*radius*sin(angle+Half_DAngle));
}
glEnd();
```

Példa

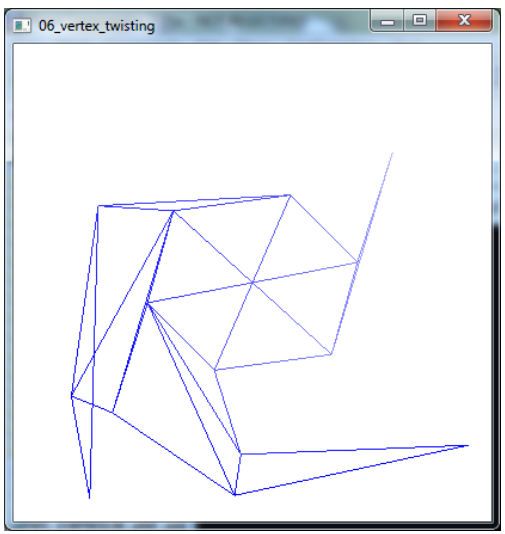
```
struct C3E4_Output {
  float4 position : POSITION;
  float4 color    : COLOR;
};
C3E4_Output C3E4v_twist(float2 position : POSITION,
                        float4 color    : COLOR,
                        uniform float twisting)
{
  C3E4_Output OUT;
  float angle = twisting * length(position);
  float cosLength, sinLength;
  sincos(angle, sinLength, cosLength);
  OUT.position[0] = cosLength * position[0] +
                    -sinLength * position[1];
  OUT.position[1] = sinLength * position[0] +
                    cosLength * position[1];
  OUT.position[2] = 0;
  OUT.position[3] = 1;
  OUT.color = color;
  return OUT;
}
```

Cg példa
A tesszalálás fontossága a vertex programoknál

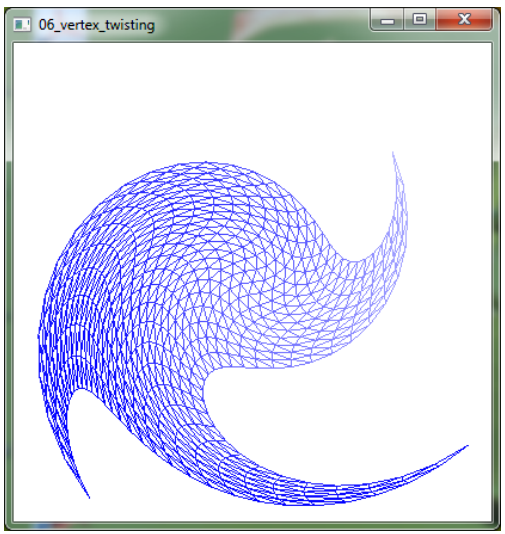
- Vertex elforgatása a középpont körül
  - A forgatási szög növelésével több vertex-re van szükség
- Általában, amikor egy vertex porgram nem lineáris számítást hajt végre, akkor megfelelő tesszalálásra van szükség az elfogadható eredmény eléréséhez



Alacsony



Közepes



Nagy

- Összetett alakzatok létrehozása OpenGL támogatással
  - OpenGL GLU segédfüggvénykönyvtár
    - Gömbök, hengerek, kúpok és sík korongok, illetve korongok lyukkal
    - Szabad-formájú felületekhez
    - Rendhagyó konkáv alakzatok, kisebb jobban kezelhető konvex alakzatokra való felbontása
- GLUT-os objektumok
  - Drótvázas/Kitöltött
  - Kocka, gömb, henger, stb.

Kvadratikus objektumok

- Másodfokú algebrai egyenletekkel leírható felületek
  - Pl. gömb, ellipszis, kúp, henger
- GLU segédfüggvény-könyvtár
  - Objektum orientált modell
    - Nagy paraméter lista elkerülése
    - Paraméterek beállítása függvényekkel
  - A felületekhez további attribútumokat/tulajdonságokat rendelhetünk
    - Normálvektorok
    - Textúra-koordináták
    - ...

- Egy üres kvadratikus objektum létrehozása, használata és törlése

```
GLUquadricObj *pObj;
// . . .
// Kvadratikus objektum létrehozása és inicializálása
pObj = gluNewQuadric();
// Renderelési paraméterek beállítása
// . . .
// Kvadratikus felület rajzolása
// . . .
// Kvadratikus objektum felszabadítása
gluDeleteQuadric(pObj);
```

- Egy üres kvadratikus objektum létrehozása, használata és törlése

```
GLUquadricObj *pObj;
// . . .
// Kvadratikus objektum létrehozása és inicializálása
pObj = gluNewQuadric();
// Renderelési paraméterek beállítása
// . . .
// Kvadratikus felület rajzolása
// . . .
// Kvadratikus objektum felszabadítása
gluDeleteQuadric(pObj);
```

- Egy üres kvadratikus objektum létrehozása, használata és törlése

```
GLUquadricObj *pObj;
// . . .
// Kvadratikus objektum létrehozása és inicializálása
pObj = gluNewQuadric();
// Renderelési paraméterek beállítása
// . . .
// Kvadratikus felület rajzolása
// . . .
// Kvadratikus objektum felszabadítása
gluDeleteQuadric(pObj);
```

- Rajzolási stílus
  - gluQuadricDrawStyle(GLUquadricObj *obj, GLenum drawStyle)

    `GLU_FILL` Solid objektumként jelenik meg
    `GLU_LINE` Drótvázas alakzatként jelenik meg
    `GLU_POINT` Vertex pontok halmazaként jelenik meg
    `GLU_SILHOUETTE` Hasonló a drótvázas megjelenéshez, de a poligonok szomszédos élei nem jelennek meg

- Normál egységvektorok automatikus generálása
  - `gluQuadricNormals(GLUquadricObj *pbj, GLenum normals)`
    - `GL_NONE` Normálvektorok nélkül
    - `GL_SMOOTH` Sima normálvektorok
      - Minden vertexhez külön-külön van meghatározva a normálvektor
    - `GL_FLAT` sík normálvektorok
      - Síkonként, adott háromszögre van kiszámítva a normálvektor

- Normálvektorok iránya
  - `gluQuadricOrientation(GLUquadricObj *obj, GLenum orientation)`

    `GLU_OUTSIDE` Kívülre mutat
    - Pl. megvilágított gömb

    `GLU_INSIDE` Belülre mutat
    - Pl. boltíves mennyezet

- Textúra-koordináták kiszámolása
  - `gluQuadricTexture(GLUquadricObj *obj, GLenum textureCoords)`
    - `GL_TRUE` Igen
      - A textúra-koordináták egyenletesen helyezkednek el a felületen
    - `GL_FALSE` Nem
      - Nem generálódnak textúra-koordináták

- Gömb
  - `gluSphere(GLUQuadricObj *obj, GLdouble radius, GLint slices, GLint stacks)`
    - `radius` A gömb sugara
    - `slices` A gyűrűkön belül lévő háromszögek/négyszögek száma
      - Hosszúsági körök száma
    - `stacks` Gyűrűk száma
      - Szélességi körök száma

- Henger
  - `gluCylinder(GLUquadricObj *obj, GLdouble baseRadius, GLdouble topRadius, GLdouble height, GLint slices, GLint stacks)`

`baseRadius`	Az origóhoz közelebbi oldalhoz tartozó sugár
`topRadius`	A másik oldalhoz tartozó sugár
`height`	A henger magassága
`slices`	Szeletek száma
`stacks`	Gyűrűk száma

  - Amennyiben a `baseRadius` vagy a `topRadius` nullával egyenlő, akkor egy kúpot kapunk eredményül

- Korong
  - `gluDisk(GLUquadricObj *obj, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops)`

    `innerRadius` Belső sugár
    - Amennyiben nullával egyenlő, akkor egy tömör korongot kapunk
    - Amennyiben nem nulla, akkor egy lyukas korong az eredmény (csavar alátét)

    `outerRadius` Külső sugár

    `slices` Szeletek száma

    `loops` Gyűrűk szűma

Bézier görbék és felületek

- Parametrikus egyenletek
  - x, y és z egy másik változó függvényeként van megadva
  - A változó egy előre definiált intervallum értékeit veheti fel
  - Egy részecske időbeli mozgása

$$x = f(t),$$
$$y = g(t),$$
$$z = h(t),$$

ahol $f(t)$, $g(t)$ és $h(t)$ egyedi függvények.

- OpenGL-be görbe esetén u-val, felület esetén u-val és v-vel jelöljük a parametrikus görbe/felület paramétereit

- A görbe definiálásakor kontroll pontokkal befolyásolhatjuk annak az alakját
  - Az első és utolsó pont része a görbének
  - A többi kontrollpont mágnesként viselkedik
    - Maguk felé húzzák a görbét
- A görbe rangját a kontrollpontok száma határozza meg
- A görbe foka eggyel kisebb, mint annak a rangja

- A kontrollpontok matematikai jelentése a görbék parametrikus polinom egyenletekre vonatkoznak

  Rang Együtthatók száma
  Fok A legnagyobb kitevője a paraméternek
  - A leggyakoribbak a kubikos (harmadfokú) görbék

- Elméletileg tetszőleges rangú görbét megadhatunk
  - magasabb rangú görbék ellenőrizhetetlenül oszcillálni kezdenek
  - A kontrollpontok kis változtatására nagy mértékben megváltoznak

- Görbék közös vég- és kezdőpontjaikban leírja, hogy mennyire sima az átmenet közöttük
  - Semmilyen: Nincs közös pontja a két görbének
  - C0: Pozícióbeli
    - Egy közös pontban találkoznak
  - C1: Érintőleges
    - A két végpontban a két görbe érintője azonos
  - C2: Görbületi
    - A görbületi sugarak is megegyeznek a töréspontban
    - Az átmenet még simább

- `void glMap1f( GLenum target, GLfloat u1, GLfloat u2, GLint stride, GLint order, const GLfloat * points);`
  - `target` A kiértékelővel előállított adat típusa
    - Pl. `GL_MAP1_VERTEX_3`, `GL_MAP1_NORMAL`, `GL_MAP1_TEXTURE_COORD_1`, ...
  - `u1, u2` Az u lineáris leképezését adja meg
  - `stride` Két kontrollpont közötti float-ok vagy double-ok száma a `points` adatstruktúrában
  - `order` A kontrollpontok száma (pozitív)
  - `points` A kontrollpontokat tartalmazó tömbre mutató pointer

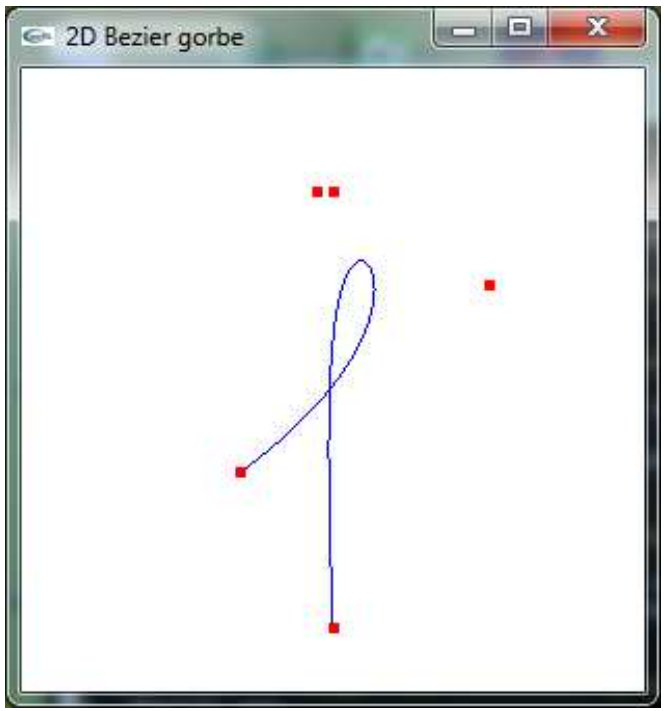
```
//A kontrollpontok száma
GLint nNumPoints = 5;

GLfloat ctrlPoints[5][3]=
  // Végpont
  {{ -3.0f, -3.0f, 0.0f},
  // Kontrollpont
  { 5.0f, 3.0f, 0.0f},
  // Kontrollpont
  { 0.0f, 6.0f, 0.0f},
  // Kontorllpont
  { -0.5f, 6.0f, 0.0f},
  // Végpont
  { 0.0f, -8.0f, 0.0f }};
```

```
glMap1f(
  // Az előállított
  // adat típusa
  GL_MAP1_VERTEX_3,
  // u alsó korlátja
  0.0f,
  // u felső korlátja
  100.0f,
  // A pontok közötti
  // távolság az adatokban
  3,
  // Kontrollpontok száma
  nNumPoints,
  // Kontrollpontokat
  // tartalmazó tömb
      mutatója
  &ctrlPoints[0][0]);
```

- `glEnable(GL_MAP1_VERTEX_3);`
  - A kiértékelő engedélyezése
- `void glEvalCoord1f(GLfloat u)`

  `u` A paraméter értékének a megadása

```
// A pontok összekötése
// töredezett vonallal
glBegin(GL_LINE_STRIP);
for(i = 0; i <= 100; i++)
{
// A görbe kiértékelése
// az adott pontban
  glEvalCoord1f((GLfloat) i
      );
}
glEnd();
```



- Egyszerűbb megvalósítás
  - `void glMapGrid1d( GLint un, GLfloat u1, GLfloat u2)`

    `un` A rács felosztása

    `u1, u2` Megadja a rácspontok leképezését.
  - `void glEvalMesh1(GLenum mode, GLint i1, GLint i2)`

    `mode` `GL_POINT` vagy `GL_LINE`

    `i1, i2` Az első és utolsó érték a tartományon

```
// Leképezi a 100 pont rácsát a 0 - 100 intervallumra
glMapGrid1d(100,0.0,100.0);

// Kiértékeli a rácsot és vonalakkal megjeleníti azt
glEvalMesh1(GL_LINE,0,100);
```

- `void glMap2f( GLenum target, GLfloat u1, GLfloat u2, GLint ustride, GLint uorder, GLfloat v1, GLfloat v2, GLint vstride, GLint vorder, const GLfloat *points)`

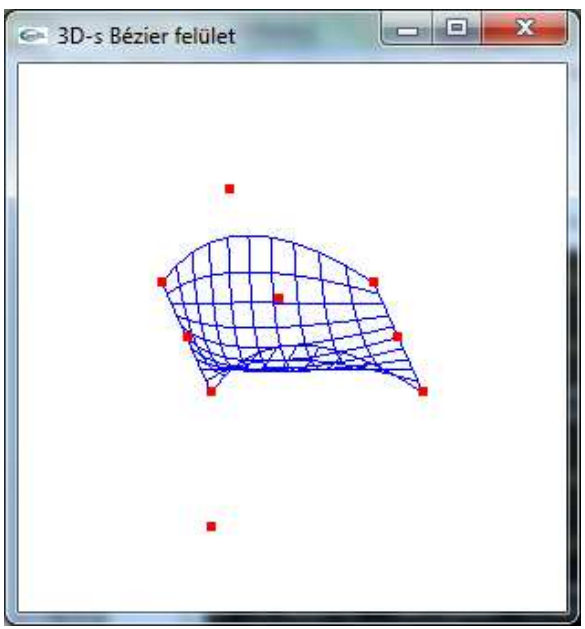
`target`	A kiértékelővel előállított adat típusa
`u1, u2`	Az u lineáris leképzését adja meg
`ustride`	Két u értelmezési tartományban lévő kontrollpont közötti float-ok vagy double-ok száma a `points` adatstruktúrában
`uorder`	A kontroll pontokat tartalmazó tömb dimenziója u tengely mentén
`v1, v2`	Az v lineáris leképzését adja meg
`vstride`	Két v értelmezési tartományban lévő kontrollpont közötti float-ok vagy double-ok száma a `points` adatstruktúrában
`vorder`	A kontroll pontokat tartalmazó tömb dimenziója v tengely mentén
`points`	Kontrollpontokat tartalmazó tömbre mutató pointer

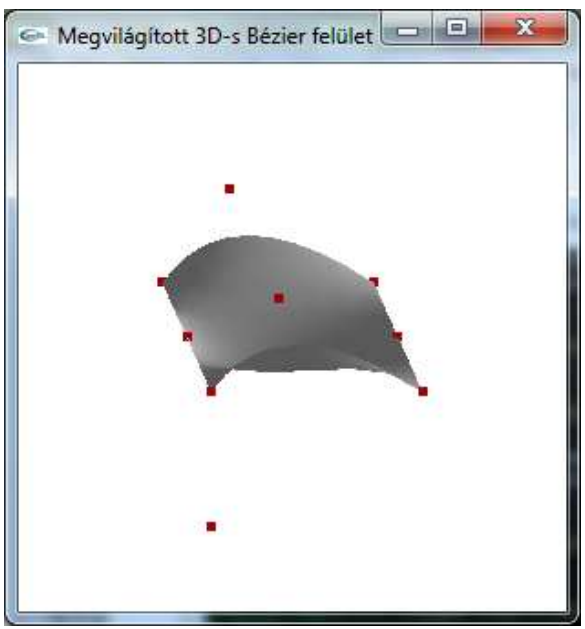
- Egyszerűbb megvalósítás
  - `void glMapGrid2f( GLint un, GLfloat u1, GLfloat u2, GLint vn, GLfloat v1, GLfloat v2 );`

    `un` A rács felosztása u irányban
    `u1, u2` Megadja a rácspontok leképezését
    `vn` A rács felosztása v irányban
    `v1, v2` Megadja a rácspontok leképezését
  - `void glEvalMesh2( GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2)`

    `mode` `GL_POINT` vagy `GL_LINE`
    `i1, i2` Az első és utolsó érték az u tartományon
    `j1, j2` Az első és utolsó érték az v tartományon

```
// Kiértékelő engedélyezése
glEnable(GL_MAP2_VERTEX_3);

// Magasabb szintű függvény a rács leképezésére
// A rács 10 pontjának a leképezése a 0 - 10 tartományra
glMapGrid2f(10, 0.0f, 10.0f, 10, 0.0f, 10.0f);

// A rács kiértékelése vonalakkal
glEvalMesh2(GL_LINE, 0, 10, 0, 10);
```





GLUT-os objektumok

- A GLUT függvénykönyvtár 3D-s geometriai objektumok létrehozására alkalmas függvények
- Normálvektorokat létrehoz
  - Nem generál textúra-koordinátákat (kivéve teáskanna)
- Pl. gömb és kúp esetén a GLUT-os megvalósítás az előbb ismertetett kvadratikus objektumokat megvalósító függvényeket használja
  - Az adott GLUT-os objektumok paraméterlistája nagyon hasonló a kvadratikus objektumokat megvalósító függvényekhez

Tömör alakzat	**Drótvázas alakzat**	**3D-s objektum**
`glutSolidSphere`	`glutWireSphere`	Gömb
`glutSolidCube`	`glutWireCube`	Kocka
`glutSolidCone`	`glutWireCone`	Kúp
`glutSolidTorus`	`glutWireTorus`	Tórusz
`glutSolidDodecahedron`	`glutWireDodecahedron`	Dodekaéder
`glutSolidOctahedron`	`glutWireOctahedron`	Oktaéder
`glutSolidTetrahedron`	`glutWireTetrahedron`	Tetraéder
`glutSolidIcosahedron`	`glutWireIcosahedron`	Ikozaéder
`glutSolidTeapot`	`glutWireTeapot`	Teáskanna

Összefoglalás

- Egy objektum felépítése
- Kvadratikus objektumok
- Bézier görbék és felületek
- GLUT-os objektumok

Megvilágítás-árnyalás, átlátszóság és köd

Árnyalás

Megvilágítás-árnyalás

- Fotorealisztikus előállítása egy háromdimenziós világnak
  - Valósághű geometriai megjelenés
  - Valósághű külső megjelenés
    - Anyagi tulajdonságok felületekhez való hozzárendelése
    - Különböző fajta fényforrások alkalmazása
    - Textúrák hozzáadása
    - Köd
    - Átlátszóság használata

- Fényforrástípusok

  Irányított fények A fényforrás az megvilágított objektumtól végtelen távoli pontban helyezkedik el

  Pontfények, reflektorfények Pozícionális fények, mind a kettő rendelkezik egy pozícióval a térben

- Mind a három fényforrás esetén megadhatunk intenzitásra vonatkozó paramétereket és szín (RGB) értékeket

Jelölés	Leírás
$\mathbf{s}_{amb}$	Ambiens intenzitás szín
$\mathbf{s}_{diff}$	Diffúz intenzitás szín
$\mathbf{s}_{spec}$	Spekuláris intenzitás szín
$\mathbf{s}_{pos}$	Négy elemű fényforrás pozíció

- További reflektorfényre vonatkozó paraméterek
  - $\mathbf{s}_{dir}$ Irányvektor
  - $\mathbf{s}_{cut}$ Levágási szög
  - $\mathbf{s}_{exp}$ Kúpon belüli elnyelődés kontrollálása
- Pozícionális fényforrások távolság alapú intenzitás vezérlésére
  - $\mathbf{s}_c, \mathbf{s}_l$ és $\mathbf{s}_q$ Csillapítás vezérlése

Anyagi tulajdonságok

- Egy felület színét az anyaghoz tartozó paraméterekkel, a fényforrások paramétereivel (melyek megvilágítják a felületet) és egy megvilágítási modellel határozhatjuk meg

Jelölés	Leírás
$\mathbf{m}_{amb}$	Ambiens anyag szín
$\mathbf{m}_{diff}$	Diffúz anyag szín
$\mathbf{m}_{spec}$	Spekuláris anyag szín
$\mathbf{m}_{shi}$	Fényesség paraméter
$\mathbf{m}_{emi}$	Emisszív anyag szín

Megvilágítás és árnyalás

Megvilágítás Megadjuk az anyag és fényforrások paramétereivel meghatározott látható szín értékeit

Árnyalás Az a folyamat, amely végrehajtja a megvilágítási számításokat és meghatározza azokból a pixelek színeit

- Flat, sík
- Goraud
- Phong

Árnyalás

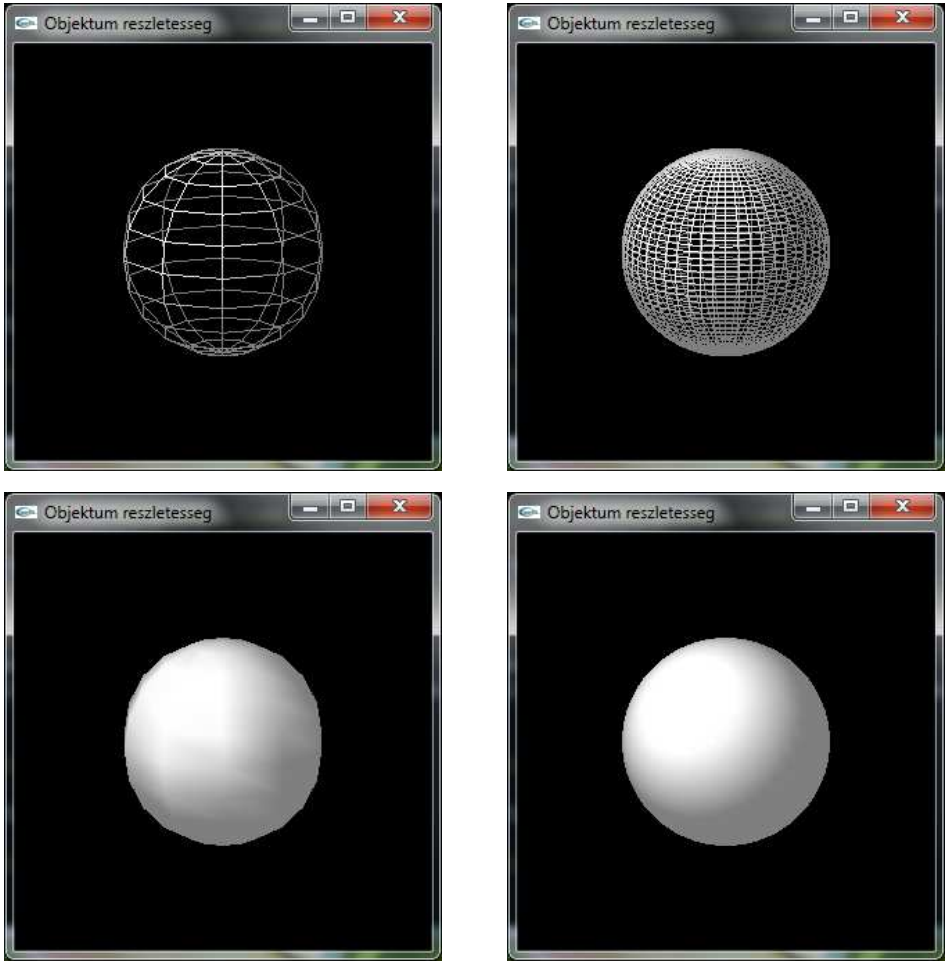
Flat A szín egy háromszögre van kiszámítva és a háromszög ezzel a színnel van kitöltve

Goraud A megvilágítás a háromszög mindegyik vertexe esetén meg van határozva és ezeket a színeket interpolálva a háromszög felületén kapjuk a végső eredményt

Phong A vertexekben tárolt árnyalási normálvektorokat interpolálva határozzuk meg a pixelenkénti normálvektorokat a háromszögben. Ezeket a normálvektorokat használva számítjuk ki a megvilágítás hatását az adott pixelben

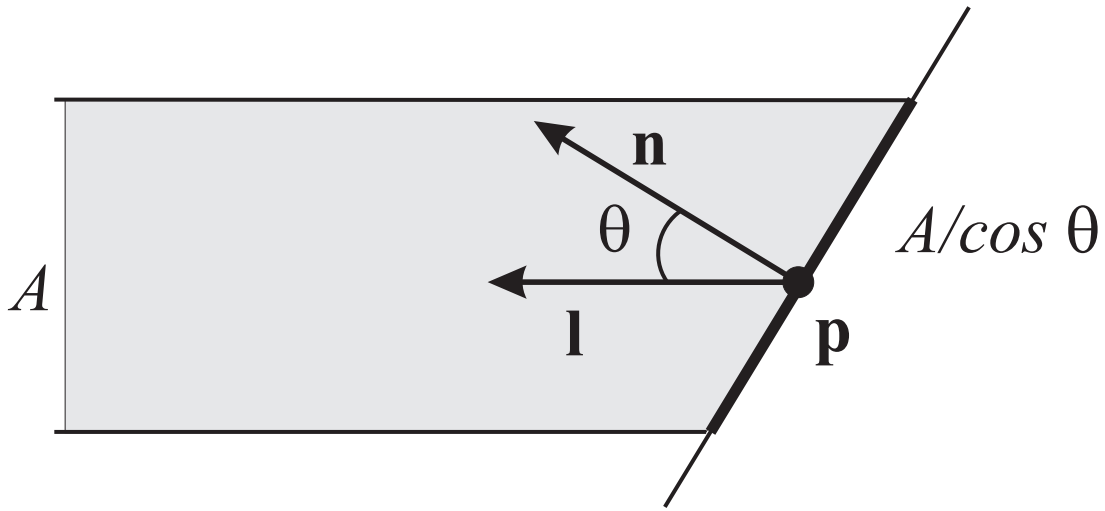
Árnyalás

- A flat árnyalást könnyű megvalósítani
  - Nem ad olyan sima eredményt görbe felület esetén
  - Meg lehet különböztetni a modellt felépítő primitíveket illetve felületeket
- A Gouraud árnyalás függ az objektum részletességétől



- Kevésbé függ az objektum kidolgozottságától
  - Felületi normálvektorok interpolálása
  - Pixelenkénti megvilágítás kiszámítása
- Kiszámítása bonyolultabb és költségesebb
- Korábban ezt a fajta módszert kevésbé használták
- Gouraud árnyalással hasonló eredményt lehet elérni
  - A felület pixelnél kisebb háromszögekre való felosztása
  - Nagyon lassú lehet
  - Nem programozható grafikus hardveren is megvalósítható

- Megfelel a fizikai valóságnak valamint a fény és felület kölcsönhatásának
- Lambert törvényen alapul
  - Az ideális diffúz (teljesen matt és nem csillogó) felületeknél a visszavert fény mértéke az **n** felületi normál és az **l** fényvektor közötti ϕ szög koszinuszától függ



$$\mathbf{i}_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\phi,$$

$\mathbf{i}_{diff}$ a szem irányában visszavert fény mértékét megadó fizikai mennyiség

$\phi > \pi/2$ esetén nullával egyenlő.

A felület a fénnyel ellentétes irányba néz

- A megvilágítási egyenlet diffúz komponense független a kamera pozíciójától és irányától
- A megvilágított felület bármely irányból ugyanúgy néz ki
- fényforrás $\mathbf{s}_{diff}$ és az anyag $\mathbf{m}_{diff}$ diffúz színét használva

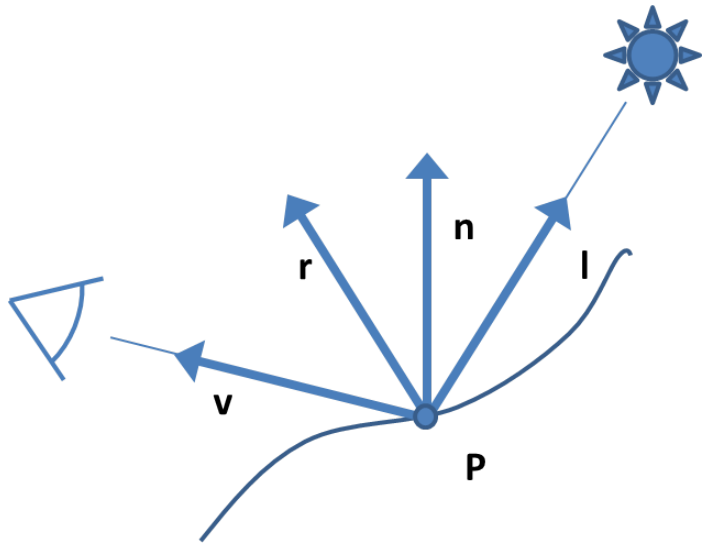
$$\mathbf{i}_{diff} = \max((\mathbf{n} \cdot \mathbf{l}), 0)\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}$$

  - Az $\mathbf{i}_{diff}$ a szín diffúz tagja
  - Az $\otimes$ operátor a komponensenkénti szorzás
  - $\max((\mathbf{n} \cdot \mathbf{l}), 0)$
    - 0, ha $\mathbf{n}$ és $\mathbf{l}$ közötti szög értéke nagyobb, mint $\pi/2$

- A spekuláris komponens a felület csillogásáért felelős
- Világos foltként jelenik meg a felületen
  - A felület görbeségét hangsúlyozza ki
  - Segít a fényforrások irányának és helyének a meghatározásában
- Phong megvilágítási egyenlet:

$$\mathbf{i}_{spec} = (\mathbf{r} \cdot \mathbf{v})^{m_{shi}} = (\cos \rho)^{m_{shi}}$$

  - **v** a **p** felületi pontból a nézőpont felé mutató vektor
  - az **r** az **l** fény vektor **n** normálvektorral meghatározott visszaverődése
  - A spekuláris összetevő annál erősebb, minél jobban egybeesik az **r** visszaverődési vektor és a **v** nézőpont vektor



- Az **l** fény vektor az **n** normálvektorra nézve az **r** vektor irányában verődik vissza
- Az **r** vektort a következőképpen lehet meghatározni:

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

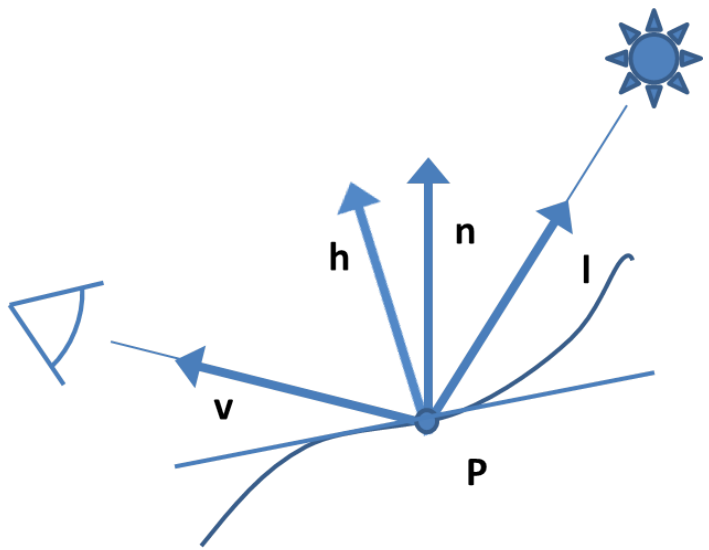
- Amennyiben $\mathbf{n} \cdot \mathbf{l} < 0$
  - A felület nem látható a fényforrásból nézve

- Blinn egyenlete

$$\mathbf{i}_{spec} = (\mathbf{n} \cdot \mathbf{h})^{m_{shi}} = (\cos\phi)^{m_{shi}}$$

  - **h** az **l** és **v** között lévő normalizált vektor

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\|\mathbf{l} + \mathbf{v}\|}$$



- A **h** annak a síknak a normálisa a **p** pontban, amely a fényforrásból tökéletesen veri vissza a fényt a nézőpontba
  - Az $\mathbf{n} \cdot \mathbf{h}$ tag akkor maximális, ha az **n** normális **p** pontban egybeesik a **h** vektorral
  - Az $\mathbf{n} \cdot \mathbf{h}$ tényező abban az esetben csökken, amikor az **n** és **h** között a szög növekszik
- Nem kell kiszámítani az **r** visszaverődési vektort
- A kétfajta spekuláris megvilágítás közötti közelítés

$$(\mathbf{r} \cdot \mathbf{v})^{m_{shi}} \approx (\mathbf{n} \cdot \mathbf{h})^{4m_{shi}}$$

- OpenGL és a Direct3D megvalósítás

$$\mathbf{i}_{spec} = \max((\mathbf{n} \cdot \mathbf{h}), 0)^{m_{shi}} \mathbf{m}_{spec} \otimes \mathbf{s}_{spec}$$

- m_{shi} a felület csillogásának a mértékét írja le
  - Értékének növelésével azt a hatást érjük el, hogy a világos terület nagysága beszűkül
- Schlick adott egy alternatív megközelítést Phong egyenletére

$$t = \cos \rho,$$

$$\mathbf{i}_{spec} = \frac{t}{m_{shi} - t m_{shi} + t} \mathbf{m}_{spec} \otimes \mathbf{s}_{spec}$$

- A megvilágítási modellünkben a fények közvetlenül ragyognak a felületeken
- A valóságban a fény a fényforrásból kiindulva egy másik felületről visszaverődve is elérheti a tárgyat
- A másik felületről érkező fény nem számítható be sem a spekuláris sem pedig a diffúz komponensbe
- Indirekt megvilágítás szimulálása
  - A megvilágítási modellbe belevesszük az ambiens tagot
  - Csak valamilyen kombinációja az anyagi és fény konstansoknak

$$\mathbf{i}_{amb} = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}$$

- Egy tárgy valamilyen minimális mennyiségű színnel fog rendelkezni
  - Még akkor is, ha nem közvetlen módon lesz megvilágítva
- Azok a felületek, melyek nem a fény felé néznek nem fognak teljesen feketén megjelenni

- OpenGL
  - Támogatja a fényforrásonkénti ambiens értéket
    - Amikor a fényt kikapcsoljuk, akkor az ambiens összetevő automatikusan el lesz távolítva
- Csak ambiens tagot használva nem kapunk megfelelő eredményt
  - Eltűnik a három-dimenziós hatás
- Mindegyik objektum meg legyen világítva legalább egy kicsi direkt megvilágítással
  - Fényeket helyezünk el a színtéren
  - Fejlámpa (headlight) használta, amely egy a nézőponthoz kapcsolt pontfény
  - A spekuláris komponensét kikapcsoljuk, hogy kevésbé zavarjon

- Lokális megvilágítási modell
  - A megvilágítás csak a fényforrásokból származó fénytől függ
  - Más felületről nem érkezik fény
- A megvilágítást az ambiens, diffúz és spekuláris komponensek határozzák meg

$$\mathbf{i}_{tot} = \mathbf{i}_{amb} + \mathbf{i}_{diff} + \mathbf{i}_{spec}$$

Cg vertex program - alap megvilágítás
Paraméterek

```
void C5E1v_basicLight(float4 position  : POSITION,
                      float3 normal    : NORMAL,

                  out float4 oPosition : POSITION,
                  out float4 color     : COLOR,

                  uniform float4x4 modelViewProj,
                  uniform float3 globalAmbient,
                  uniform float3 lightColor,
                  uniform float3 lightPosition,
                  uniform float3 eyePosition,
                  uniform float3 Ka,
                  uniform float3 Kd,
                  uniform float3 Ks,
                  uniform float  shininess)
```

```
oPosition = mul(modelViewProj, position);

float3 P = position.xyz;
float3 N = normal;
```

```
// Ambiens tag
float3 ambient = Ka * globalAmbient;

// Diffúz tag
float3 L = normalize(lightPosition - P);
float diffuseLight = max(dot(N, L), 0);
float3 diffuse = Kd * lightColor * diffuseLight;
```

Cg vertex program - alap megvilágítás
Spekuláris tag kiszámítása

```
float3 V = normalize(eyePosition - P);
float3 H = normalize(L + V);

float specularLight =
    pow(max(dot(N, H), 0), shininess);

if (diffuseLight <= 0)
    specularLight = 0;

float3 specular =
    Ks * lightColor * specularLight;
```

```
color.xyz = ambient + diffuse + specular;
color.w = 1;
```

Cg fragmens program - alap megvilágítás
Vertex program

```
void C5E2v_fragmentLighting(float4 position : POSITION,
                  float3 normal   : NORMAL,

                  loat4 oPosition : POSITION,
                  out float3 objectPos : TEXCOORD0,
                  out float3 oNormal   : TEXCOORD1,
                  uniform float4x4 modelViewProj)

  oPosition = mul(modelViewProj, position);

  objectPos = position.xyz;
  oNormal = normal;
```

Cg fragmens program - alap megvilágítás

Vertex program

```
void C5E2v_fragmentLighting(float4 position : POSITION,
                  float3 normal   : NORMAL,

                  loat4 oPosition : POSITION,
                  out float3 objectPos : TEXCOORD0,
                  out float3 oNormal   : TEXCOORD1,
                  uniform float4x4 modelViewProj)

  oPosition = mul(modelViewProj, position);

  objectPos = position.xyz;
  oNormal = normal;
```

Cg fragmens program - alap megvilágítás

Vertex program

```
void C5E2v_fragmentLighting(float4 position : POSITION,
                  float3 normal   : NORMAL,

                  loat4 oPosition : POSITION,
                  out float3 objectPos : TEXCOORD0,
                  out float3 oNormal   : TEXCOORD1,
                  uniform float4x4 modelViewProj)

  oPosition = mul(modelViewProj, position);

  objectPos = position.xyz;
  oNormal = normal;
```

Cg fragmens program - alap megvilágítás
Paraméterek

```
void C5E3f_basicLight(float4 position  : TEXCOORD0,
                      float3 normal    : TEXCOORD1,

                  out float4 color     : COLOR,

                  uniform float3 globalAmbient,
                  uniform float3 lightColor,
                  uniform float3 lightPosition,
                  uniform float3 eyePosition,
                  uniform float3 Ka,
                  uniform float3 Kd,
                  uniform float3 Ks,
                  uniform float  shininess)
```

Cg fragmens program - alap megvilágítás
Paraméterek

```
void C5E3f_basicLight(float4 position  : TEXCOORD0,
                      float3 normal    : TEXCOORD1,

                  out float4 color     : COLOR,

                  uniform float3 globalAmbient,
                  uniform float3 lightColor,
                  uniform float3 lightPosition,
                  uniform float3 eyePosition,
                  uniform float3 Ka,
                  uniform float3 Kd,
                  uniform float3 Ks,
                  uniform float  shininess)
```

Cg fragmens program - alap megvilágítás
Paraméterek

```
void C5E3f_basicLight(float4 position  : TEXCOORD0,
                      float3 normal    : TEXCOORD1,

                  out float4 color     : COLOR,

                  uniform float3 globalAmbient,
                  uniform float3 lightColor,
                  uniform float3 lightPosition,
                  uniform float3 eyePosition,
                  uniform float3 Ka,
                  uniform float3 Kd,
                  uniform float3 Ks,
                  uniform float  shininess)
```

```
float3 P = position.xyz;
float3 N = normalize(normal);

// Ambiens tag
float3 ambient = Ka * globalAmbient;

// Diffúz tag
float3 L = normalize(lightPosition - P);
float diffuseLight = max(dot(L, N), 0);
float3 diffuse = Kd * lightColor * diffuseLight;

// Spekuláris tag
float3 V = normalize(eyePosition - P);
float3 H = normalize(L + V);
float specularLight = pow(max(dot(H, N), 0), shininess);
if (diffuseLight <= 0) specularLight = 0;
float3 specular = Ks * lightColor * specularLight;

color.xyz =  ambient + diffuse + specular;
color.w = 1;
```

- A valóságban a fény intenzitása fordítottan arányos a fényforrástól mért távolság négyzetével

$$d = \frac{1}{s_c + s_l \left\|\mathbf{s}_{pos} - \mathbf{p}\right\| + s_q \left\|\mathbf{s}_{pos} - \mathbf{p}\right\|^2}$$

  - $\left\|\mathbf{s}_{pos} - \mathbf{p}\right\|$ az $\mathbf{s}_{pos}$ fényforrás pozíciójától vett távolság a $\mathbf{p}$ pontig
  - s_c a konstans, az s_l a lineáris és a s_q a kvadratikus csillapítást kontrollálják
- A fizikailag korrekt távolság csillapításhoz
  - $s_c = 0$, $s_l = 0$ és $s_q = 1$

$$\mathbf{i}_{tot} = \mathbf{i}_{amb} + d(\mathbf{i}_{diff} + \mathbf{i}_{spec})$$

Cg fragmens program - távolság függés

```
// Anyagi tulajdonságok
struct Material {
  float3 Ka;
  float3 Kd;
  float3 Ks;
  float shininess;
};

// Fény paraméterek
struct Light {
  float3 position;
  float3 color;
  float kC, kL, kQ;
};

// Távolságtól függő skalár meghatározása
float C5E6_attenuation(float3 P,
                       Light light)
{
  float d = distance(P, light.position);
  return 1 / (light.kC + light.kL * d + light.kQ * d * d);
}
```

```
void C5E7_attenuateLighting(Light light,
                            float3 P,
                            float3 N,
                            float3 eyePosition,
                            float  shininess,

                            out float3 diffuseResult,
                            out float3 specularResult)
```

Cg fragmens program - távolság függés

Függvénytörzs

```
// Elnyelődés kiszámítása
float attenuation = C5E6_attenuation(P, light);

// Diffúz komponens kiszámítása
float3 L = normalize(light.position - P);
float diffuseLight = max(dot(L, N), 0);
diffuseResult = attenuation *
                light.color * diffuseLight;

// Spekuláris komponens kiszámítása
float3 V = normalize(eyePosition - P);
float3 H = normalize(L + V);
float specularLight = pow(max(dot(H, N), 0),
                          shininess);
if (diffuseLight <= 0) specularLight = 0;
specularResult = attenuation *
                 light.color * specularLight;
```

Cg fragmens program - távolság függés

Belépő függvény - paraméterek

```
void oneLight(float4 position : TEXCOORD0,
              float3 normal   : TEXCOORD1,

      out float4 color      : COLOR,

      uniform float3   eyePosition,
      uniform float3   globalAmbient,
      uniform Light    lights[1],
      uniform Material material)
```

Cg fragmens program - távolság függés

Belépő függvény

```
// Ambiens
float3 ambient = material.Ka * globalAmbient;


float3 diffuseLight;
float3 specularLight;
float3 diffuseSum  = 0;
float3 specularSum = 0;

//Diffúz és spekuláris komponensek
//távolság függő kiszámítása
C5E7_attenuateLighting(lights[0], position.xyz, normal,
                       eyePosition, material.shininess,
                       diffuseLight, specularLight);
diffuseSum += diffuseLight;
specularSum += specularLight;

// Anyagi tulajdonságok figyelembevétele
float3 diffuse = material.Kd * diffuseSum;
float3 specular = material.Ks * specularSum;

color.xyz =  ambient + diffuse + specular;
color.w = 1;
```

- A reflektorfény a színteret különböző módon világítja meg
  - c_{spot}-tal jelölt szorzótényező

$$c_{spot} = \max(-\mathbf{l} \cdot \mathbf{s}_{dir}, 0)^{s_{exp}}$$

$\mathbf{l}$	A fény vektor
$\mathbf{s}_{dir}$	A reflektor iránya
s_{exp}	Az exponenciális faktor a reflektor középpontjától való halványodását vezérli

- Módosított megvilágítási egyenlet

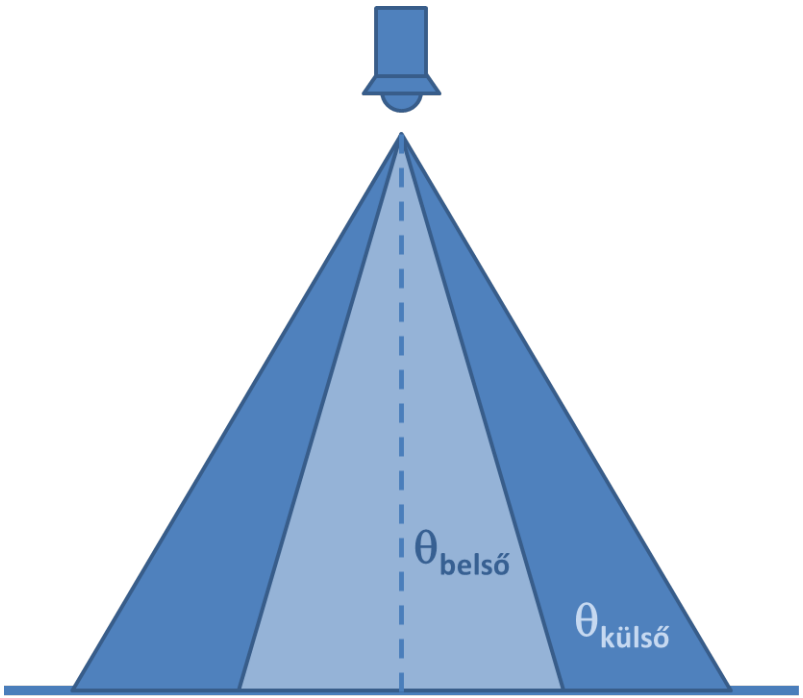
$$\mathbf{i}_{tot} = c_{spot}(\mathbf{i}_{amb} + d(\mathbf{i}_{diff} + \mathbf{i}_{spec}))$$

  - Ha a fényforrásunk nem reflektorfény, akkor $c_{spot} = 1$

Cg fragmens program - reflektorfény

Intenzitás változás

- Belső és külső "kúpok"
  - Könnyű eldönteni, hogy melyik részbe esik egy pont
  - Változtatni kell az intenzitás kiszámítását



Cg fragmens program - reflektorfény

Intenzitás változás

```
float saturate(float x)
{
  return max(0, min(1, x));
}

float smoothstep(float min,
                 float max,
                 float x)
{
  float t =
  saturate((x - min)/(max -
      min));
  return t*t*(3.0 - (2.0*t));
}
```

- 0, ha $x < min$
- 1, ha $x > max$
- Hermite interpolált érték 0 és 1 között
  - $-2t^3 + 3t^2$

Cg fragmens program - reflektorfény

Intenzitás változás

```
float C5E9_dualConeSpotlight(float3 P,
                             Light  light)

  float3 V = normalize(P - light.position);
  float cosOuterCone = light.cosOuterCone;
  float cosInnerCone = light.cosInnerCone;
  float cosDirection = dot(V, light.direction);

  return smoothstep(cosOuterCone,
                    cosInnerCone,
                    cosDirection);
```

Cg fragmens program - reflektorfény

Intenzitás változás

```
float C5E9_dualConeSpotlight(float3 P,
                             Light  light)

  float3 V = normalize(P - light.position);
  float cosOuterCone = light.cosOuterCone;
  float cosInnerCone = light.cosInnerCone;
  float cosDirection = dot(V, light.direction);

  return smoothstep(cosOuterCone,
                    cosInnerCone,
                    cosDirection);
```

- A felület mennyi fényt bocsát ki
  - Az anyag rendelkezik egy $\mathbf{m}_{emi}$ emisszív paraméterrel
- Globális ambiens fényforrás paraméter
  - Konstans háttérfényt közelít
  - Minden irányból körülveszi a tárgyakat
- Módosított megvilágítási egyenlet

$$\mathbf{i}_{tot} = \mathbf{a}_{glob} \otimes \mathbf{m}_{amb} + \mathbf{m}_{emi} + c_{spot}(\mathbf{i}_{amb} + d(\mathbf{i}_{diff} + \mathbf{i}_{spec}))$$

Cg fragmens program részlet
Emisszív paraméter

```
struct Material
  float3 Ke;
  float3 Ka;
  float3 Kd;
  float3 Ks;
  float shininess;
;

...

float3 emissive = material.Ke;
```

Cg fragmens program részlet
Emisszív paraméter

```
struct Material
  float3 Ke;
  float3 Ka;
  float3 Kd;
  float3 Ks;
  float shininess;
;

...

float3 emissive = material.Ke;
```

- Tegyük fel, hogy n fényforrásunk van és mindegyiket k indexszel azonosítjuk

$$\mathbf{i}_{tot} = \mathbf{a}_{glob} \otimes \mathbf{m}_{amb} + \mathbf{m}_{emi} + \sum_{k=1}^{n} c^k_{spot}(\mathbf{i}^k_{amb} + d^k(\mathbf{i}^k_{diff} + \mathbf{i}^k_{spec}))$$

  - A fényforrás intenzitás összege 1-nél nagyobb is lehet
    - Az eredmény megvilágítási színt $[0, 1]$ intervallumra korlátozzuk le
  - Túlcsorduló szín skálázása a legnagyobb komponenssel
  - A túlcsordulások gyakran a geometriai részletességet csökkentik

Cg fragmens program részlet
Több fényforrás

```
    float3 diffuseLight;
    float3 specularLight;

    float3 ambientSum  = 0;
    float3 diffuseSum  = 0;
    float3 specularSum = 0;

// Az ambiens, diffúz és spekuláris komponensekre való
     számítások
    for (int i = 0; i < 2; i++) {
      C5E5_computeLighting(lights[i], position.xyz, normal,
                           eyePosition, material.shininess,
                           diffuseLight, specularLight);
      ambientSum += ambientLight;
      diffuseSum += diffuseLight;
      specularSum += specularLight;
    }

    // Anyagi tulajdonságok figyelembevétele
    float3 amibient = material.Ka * ambientSum;
    float3 diffuse = material.Kd * diffuseSum;
    float3 specular = material.Ks * specularSum;
```

Átlátszóság

Átlátszóság

- Megvalósításához szükség van az átlátszó tárgy színének és a mögötte lévő objektumok színének a keverésére
- Egy RGB szín és egy Z-puffer mélység van hozzákötve mindegyik pixelhez a képernyőn való megjelenítésekor
- α komponens
  - Az az érték, amely leírja a tárgy átlátszóságának a fokát egy adott pixelben
  - $\alpha = 1$ azt jelenti, hogy az objektum nem átlátszó és teljes egészében kitölti a pixel területet
  - $\alpha = 0$ pedig azt jelenti, hogy a pixel egyáltalán nem látszik

Átlátszóság

- Egy objektum átlátszóvá tételéhez a meglévő színtéren kell megjeleníteni egynél kisebb alfa értékkel

$$\mathbf{c}_o = \alpha_s \mathbf{c}_s + (1 - \alpha_s)\mathbf{c}_d \qquad \text{[over operátor]}$$

- $\mathbf{c}_s$ Az átlátszó objektum színe (forrás)
- α_s A tárgy alfa értéke
- $\mathbf{c}_d$ A keveredés előtti (a színpufferben lévő, cél) pixel szín érték
- $\mathbf{c}_o$ Az eredmény szín
  - Az átlátszó objektumot a meglévő színtér elé (over) helyezzük

Átlátszóság

- Helyes megjelenítéséhez általában szükségünk van rendezésre
  - Először a nem átlátszó tárgyakat kell renderelni
  - Aztán az átlátszó objektumokat kell hátulról előre haladva összekeverni a háttérben lévő alakzatok pixel értékeivel
- Tetszőleges sorrendben való összekeverés esetén súlyos artifaktumokat kaphatunk
  - A művelet sorrendfüggő vagyis feltételezi, hogy a háttérben lévő tárgyak már a színpufferben vannak
  - Speciális esetben, amikor két átlátszó tárgy van megjelenítve és mind a kettő alfa értéke 0.5, akkor a keveredésnél nem számít a sorrend

Átlátszóság

- Amennyiben a rendezés nem lehetséges vagy csak részben lett végrehajtva
  - Legjobb a Z-puffer használata
    - A z-mélység írását kikapcsolva az átlátszó objektum esetén
    - Az összes átlátszó objektum legalább meg fog jelenni
- Más technikák
  - Hátsó oldalak eldobásának kikapcsolása
  - Az átlátszó poligonok kétszeri renderelésével és a mélység tesztelést valamint a Z-puffer írásának az engedélyezését váltogatva

Átlátszóság

- Ki lehet számítani több menetben, két vagy több mélységpuffer használatával (mélység hámozás)
  - Első megjelenítési menetben a nem átlátszó felületek z-mélység értékeit helyezzük el az első Z-pufferben
  - Ezután az átlátszó objektumokat rendereljük le
    - A második menetben a mélység tesztet úgy módosítjuk, hogy elfogadjuk azt a felületet, amely az első pufferben lévő z-mélység értéknél közelebb van és az átlátszó objektumok közül pedig a legtávolabb van
    - A legtávolabbi átlátszó objektum bekerül a színpufferbe a mélység értéke pedig a második Z-pufferbe
    - Ezt a puffert aztán arra használjuk, hogy a következő legközelebbi átlátszó felületet határozzuk meg a következő menetben és így tovább

Köd

Köd

- A ködöt több céllal is lehet használni
  - A külső tér realisztikusabb megjelenítés szintjének a növelése
  - Mivel a köd hatása a nézőponttól távolodva növekszik, ezért ez segít meghatározni, hogy milyen távol találhatóak az objektumok
  - Ha megfelelően használjuk, akkor ez segít a távoli vágósík hatásának az elrejtésében
  - A köd gyakran hardveresen van megvalósítva, így egy elhanyagolható plusz költséggel lehet azt használni.

- $\mathbf{c}_p$ végső pixel szín értékének meghatározása

$$\mathbf{c}_p = f\mathbf{c}_s + (1 - f)\mathbf{c}_f$$

$\mathbf{c}_f$ A köd színe
$f \in \lfloor 0, 1 \rfloor$ A köd együtthatója
$\mathbf{c}_s$ Az árnyalt oldal színe

- Ahogy f értéke csökken, a köd hatása növekszik
- Különböző egyenleteket használhatunk a f megadására

- Egy köd konstans
  - Lineárisan csökken a nézőponttól távolodva
- Hol kezdődik és hol végződik a köd a néző z-tengelye mentén?

$$f = \frac{z_{end} - z_p}{z_{end} - z_{start}}$$

  - z_p az a z érték, ahol a köd hatását kell meghatározni

- f ködegyüttható

$$f = e^{-d_f z_p},$$
$$f = e^{(-d_f z_p)^2}$$

d_f A köd sűrűségét vezérli

- A kapott értéket a $[0, 1]$ intervallumra csonkoljuk és a köd egyenletét használjuk a végső érték kiszámításához
  - $\mathbf{c}_p = f\mathbf{c}_s + (1 - f)\mathbf{c}_f$

- Néha táblázatokat használnak a ködfüggvény hardveres megvalósítása esetén
  - Minden mélységre egy f ködegyütthatót előre kiszámítanak és eltárolnak.
  - Kiolvassák a táblázatból (vagy lineáris interpolációval határozzák meg két szomszédos tábla elemből)
    - Bármilyen értéket el lehet helyezni a köd táblázatban, nem csak az iménti egyenletekben megadottakat

Köd

- A ködfüggvényeket alkalmazni lehet vertex vagy pixel szinten

  Vertex-szintű A köd hatása a megvilágítási egyenlet részeként lesz kiszámítva és a kiszámított szín értéket interpolálja a poligonon keresztül Gouraud árnyalást használva

  Pixel-szintű A pixelenként tárolt mélység értéket használva számítjuk ki

- A pixel-szintű köd jobb eredményt ad

Összefoglalás

Témakörök

- Megvilágítás
  - Fényforrások és anyagi tulajdonságok
- Árnyalás
  - Megvilágítási számítások
- Megvilágítási modell
  - Cg példaprogramok
- Átlátszóság
- Köd