RTG-formatet

2004-08-11

Gustav Taxén, gustavt@nada.kth.se

Det här texten beskriver kortfattat RTG-filformatet och ett C-bibliotek som kan läsa det. RTG-formatet är ett filformat för geometribeskrivningar som kan exporteras från t.ex. Maya. Här är ett exempel på hur det kan se ut:

```
HEADER_TITLE Alias|Wavefront Rtg Output
HEADER_VERSION 4.04

NUMBER_OF_OBJECTS 2

OUTPUT_VERT_NORMS    on
OUTPUT_TEX_COORDS    on
...
```

HUVUD

```
MATERIAL_LIST 1
    MATERIAL 0
    NAME lambert1
    AMBIENT 0.500000 0.500000 0.500000
    ...

END_MATERIAL_LIST
```

MATERIAL-LISTA

```
HIERARCHY_LIST HX 1 top_level
0 P box0
    tran: 0.000000 0.000000 0.000000
    rot: 0.174533 0.349066 0.523599
    scal: 1.000000 1.000000 1.000000

    1 P box1
        tran: 0.000000 -2.910448 0.000000
        rot: 0.174533 0.349066 0.523599
        scal: 1.000000 1.000000 1.000000

END_HIERARCHY_LIST
```

HIERARKI-BESKRIVNING

```
OBJECT_START box0 v29 n96 t41 p21
USES_MATERIAL 0 lambert1

VERTEX local
  -0.500000 -0.500000 0.500000
  0.500000 -0.500000 0.500000
  ...

NORMAL
  0.000000 0.000000 1.000000
  0.000000 0.000000 1.000000
  ...

TEXCOORD
  0.000000 0.000000
  1.000000 0.000000
  ...

POLYGON
v 0 1 2 3 4 5 6 n 0 1 2 3 4 5 6 t 0 1 2 3 4 5 6
v 7 8 9 10 n 7 8 9 10 t 7 8 9 10
...

OBJECT_END
...
```

OBJEKT-BESKRIVNING

Alla RTG-filer innehåller ett *huvud med flaggor*, en *materiallista* (som kan vara tom), en *hierarkibeskrivning* (med minst en toppnivå-nod) och, för varje nod i hierarkin, en *objektbeskrivning*. (Det kan också ingå andra saker, men de här fyra är de mest väsentliga.)

Flaggorna beskriver vilken data som finns i filen, bl.a. om polygonhörnen har normal och texturkoordinater.

Materiallistan är, som namnet antyder, helt enkelt en lista över de material som används av geometriobjekten i filen. Varje material har ett namn (samma namn som det tilldelats i Maya), parametrar för ljusreflektion (ambient, diffuse, specular, shininess, emission och transparency) och eventuellt en textur. En textur består av ett namn (samma namn som i Maya, kan vara den tomma strängen) och en fil (den fil som laddades i Maya).

NOT: Om man inte skapat sitt Maya-projekt på ”rätt” sätt (se Mayalabbpeket) så kan det hända att texturfilnamnet får en absolut sökväg istället för en relativ. Då kan man bli tvungen att gå in och ändra i RTG-filen för hand så att det stämmer.

Efter materialen följer en specifikation av de objekt som ingår. Den här hierarkin är väsentligen densamma som i Outliner-fönstret i Maya. Varje nod är antingen en geometrinod eller en grupp. Grupper saknar geometrisk information och finns därför inte med i objektbeskrivningen som följer efter hierarkibeskrivningen. Om Mayas RTG-exporter är inställd så att den exporterar transformationer (se Mayalabbpeket) så har också varje nod en transformation: translation (`tran`), rotation (`rot`) och skalning (`scal`). Eftersom kombinationer av transformationer inte är kommutativa är ordningen som man applicerar på dem viktig. Här är ”korrekt” ordning för OpenGL:

```
glTranslatef(tran.x, tran.y, tran.z);
glRotatef(RADIANS_TO_DEGREES * rot.x, 1, 0, 0);
glRotatef(RADIANS_TO_DEGREES * rot.y, 0, 1, 0);
glRotatef(RADIANS_TO_DEGREES * rot.z, 0, 0, 1);
glScalef(scal.x, scal.y, scal.z);
```

NOT: Observera att varje transformation är i förhållande till föräldernoden.

Efter hierarkibeskrivningen följer geometriska data för alla noder som inte är gruppnoder. Varje sådant avsnitt börjar med nodens namn (samma som i hierarkibeskrivningen, ”`box0`” i exemplet på förra sidan) och antalet positioner, normaler, texturkoordinater och polygoner. Sedan listas alla positioner (`VERTEX`), normaler (`NORMAL`) och texturkoordinater (`TEXCOORD`). Därefter följer polygonerna; varje polygon indexerar listorna (”`v`” för positioner, ”`n`” för normaler och ”`t`” för texturkoordinater). I exemplet ovan är alltså den första polygonen en sjuhörning vars första hörn sitter i positionen [-0.5, -0.5, 0.5], med normal [0, 0, 1] och texturkoordinat [0, 0]. Den andra polygonen är en fyrhörning.

I katalogen `/info/agi`*nn*`/OpenGL` där *nn* är nuvarande läsår finns ett bibliotek som kan tolka RTG-filer. Biblioteket innehåller två funktioner, `RTG_Parse()` och `RTG_Free()` samt en mindre mängd datastrukturer. `RTG_Parse()` används för att ladda en RTG-fil och `RTG_Free()` används för att släppa det minne som upptas av de datastrukturer som `RTG_Parse()` skapar. Så här används de:

```
#include <RTGlib.h>
...
RTGFile *file = RTG_Parse(”minfil.rtg”, 1);
...
RTGFree(file);
```

Ettan som skickas med till `RTG_Parse()` talar om att statusinformation ska skrivas ut under tiden filen tolkas. Om du inte vill se statusinformationen, skicka med en nolla istället.

Datastrukturerna i RTG-biblioteket ser ut så här:

```
/* RTG-fil med lista med toppnivånoder och lista med material */
struct RTGFile {
      RTGNode *nodes[];     // Lista med noder
      unsigned int nnodes;  // Antal noder
      RTGMaterial *mat[];   // Lista med material
      unsigned int nmat;    // Antal material
 };

/* En RTG-nod med barn, ev. geometridata och transformationsinfo */
struct RTGNode {
      struct RTGNode *children[];  // Lista med barn till denna nod
      unsigned int nchildren;      // Antal barn
      RTGObject *obj;              // Geometriinfo (kan vara NULL)
      char *objName;               // Nodens namn
      RTGVec3 trn;                 // Translation
      RTGVec3 rot;                 // Rotation
      RTGVec3 scl;                 // Skalning
 };

/* Tredimensionell vektor */
struct RTGVec3 {
      float x;
      float y;
      float z;
};

/* Tvådimensionell vektor */
struct RTGVec2 {
      float x;
      float y;
};
```

```
/* Geometridata med materialindex, positioner,
 * normaler, texturkoordinater och polygoner */
struct RTGObject {
      char *name;
      unsigned int mat;   // Index för materialet
      RTGVec3 *pos;       // Lista med positioner
      unsigned int npos;  // Antal positioner
      RTGVec3 *norm;      // Lista med normaler
      unsigned int nnorm; // Antal normaler
      RTGVec2 *tex;       // Lista med texturkoordinater
      unsigned int ntex;  // Antal texturkoordinater
      RTGPolygon *poly[]; // Lista med polygoner
      unsigned int npoly; // Antal polygoner
};

/* En polygon med en lista med hörn */
struct RTGPolygon {
      RTGVertex *v;         // Lista med hörn
      unsigned int nvert;   // Antal hörn
};

/* Polygonhörn med index till positon-, normal- och
 * texturkoordinatslistorna */
struct RTGVertex {
      unsigned int p;  // Index för positionen
      unsigned int n;  // Index för normalen
      unsigned int t;  // Index för texturkoordinaten
};

/* Ett material med namn, reflektionsegenskaper och texturinfo */
struct RTGMaterial {
      char *name;    // Materialets namn
      float amb[3];  // Ambient
      float dif[3];  // Diffuse
      float spc[3];  // Specular
      float shn;     // Shininess (cosinus-exponent i Phong-modellen)
      float em[3];   // Emission ("incandescence" i Maya)
      float trp;     // Transparency
      char *texName; // Texturnamn (kan vara NULL)
      char *texFile; // Texturfil (kan vara NULL)
};
```

Som man kan vänta sig är mappningen mellan Maya-shaders och RTG-material icketrivial. Mayas Phong-material är att föredra (ger mest lika resultat), medan Blinn, Lambert och de andra materialtyperna kan ge oväntade mappningar (eller inte funka alls).

Om Maya-materialet har en textur sätter Mayas RTG-exporter `dif` till [0, 0, 0]. Logiken bakom detta är att diffuse-värdena i Maya ”ersätts” av texturvärdena. Tyvärr funkar inte detta tänk i OpenGL där man oftast vill använda `GL_MODULATE` som parameter till `glTexEnv()`, så RTG-biblioteket sätter `dif` till [1, 1, 1] då materialet har textur.

Emission-parametern i RTG-filen är kopplad till Maya-materialens incandescence-parameter och har inget med Maya-ljuskällor att göra.

Att gå igenom en `RTGFile`-struktur som lästs in med `RTG_Parse()` är inte särskilt komplicerat (kodsnutten finns som textfil i `/info/agi`-katalogen):

```
void draw(RTGFile *file) {
      unsigned int i;
      // Rita alla toppnivånoder
      for (i = 0; i < file->nnodes; i++) {
            drawNode(file->nodes[i]);
      }
}

void drawNode(RTGFile *file, RTGNode *node) {
      unsigned int i;
      glPushMatrix();
      applicera transformation (se ovan);
      // Om noden har geometri, rita den
      if (node->obj) {
            drawObject(file, node->obj);
      }
      // Om noden har barn, rita dem
      for (i = 0; i < node->nchildren; i++) {
            drawNode(file, node->children[i]);
      }
      glPopMatrix();
}

void drawObject(RTGFile *file, RTGObject *obj) {
      unsigned int i, j;
      RTGVertex *v;
      RTGVec3 *n;
      RTGVec3 *p;
      RTGVec2 *t;
      setMaterial(file->mat[obj->mat]);
      for (i = 0; i < obj->npoly; i++) {
            glBegin(GL_POLYGON);
            for (j = 0; j < obj->poly[i]->nvert; j++) {
                  v = &obj->poly[i]->v[j];
                  n = &obj->norm[v->n];
                  t = &obj->tex[v->t];
                  p = &obj->pos[v->p];
                  glNormal3f(n->x, n->y, n->z);
                  glTexCoord2f(t->x, t->y);
                  glVertex3f(p->x, p->y, p->z);
            }
            glEnd();
      }
}

void setMaterial(RTGMaterial *mat) {
      ...
}
```