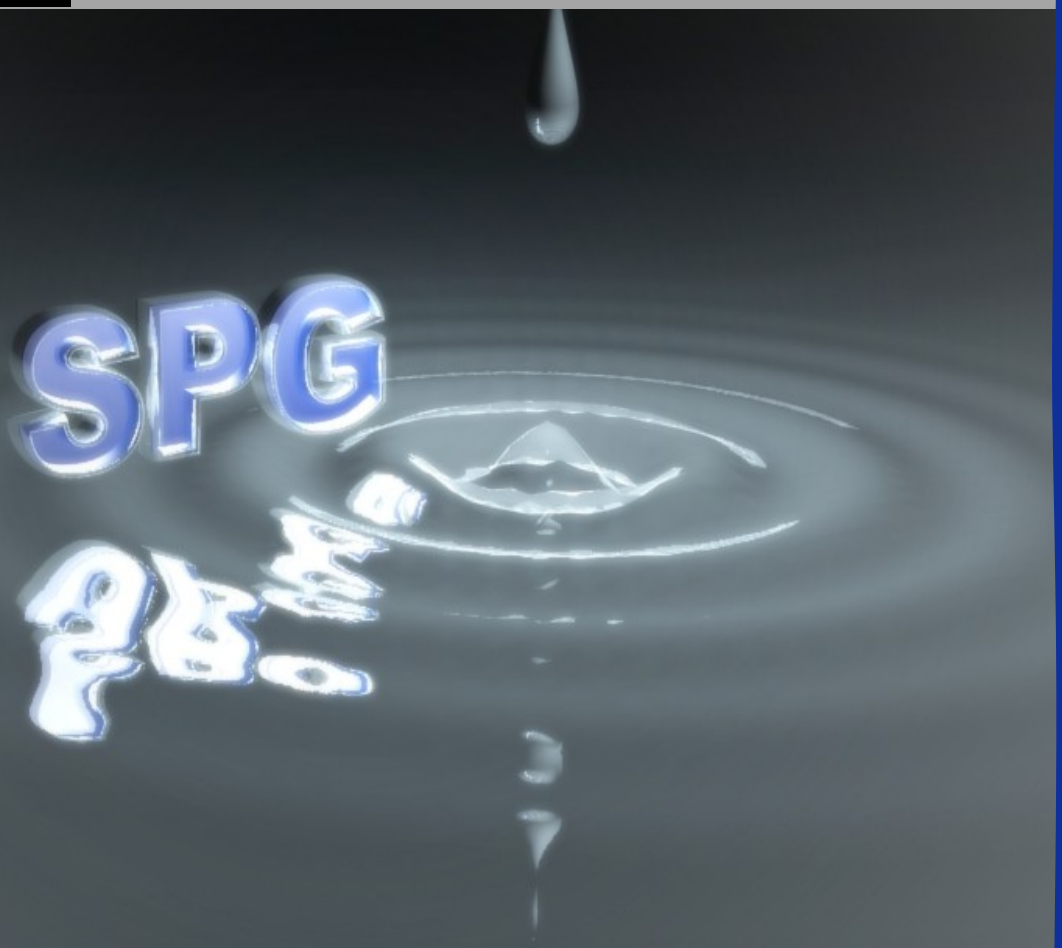
26/10/2015

FACULTATEA DE AUTOMATICA SI CALCULATOARE

SISTEME DE PRELUCRARE GRAFICA



Laborator 3

Reflexii si Framebuffere

Introducere

In acest laborator vom introduce elemente noi de OpenGL cat si un algoritm pentru calcularea reflexiilor. Cum rezolvarea corecta a problemei reflexiilor este o problema extrem de complicata in banda de rasterizare, vom introduce un algoritm primitiv ce ne ve oferi un rezultat orientativ (deci nu matematic corect), suficient pentru a intelege a reprezenta efectul de reflectie.

Framebuffere

Pana in momentul de fata am folosit top timpul framebuffer-ul default oferit de freeglut. Desi am avut oportunitatea de a selecta intre mai multe configuratii de buffere (culoare, adancime, stencil, msaa) nu am avut capacitatea de ne defini manual acest framebuffer. In acest laborator vom invata sa lucram cu framebuffere si sa le folosim pentru a implementa algoritmul render to texture.

Pentru a crea un obiect tip framebuffer putem folosi comanda:

```
unsigned int framebuffer_object;
glGenFramebuffers(1, &framebuffer_object);
```

iar pentru a lega framebufferul nou creat la banda grafica (in locul celui default), folosim:

```
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer_object);
```

In momentul de fata avem un framebuffer nou, gol (nu are nici un buffer atasat) dar care este legat la banda grafica. Evident nu ne dorim un framebuffer gol pentru ca scopul unui framebuffer e de a retine date. Putem lega texturi la framebuffer si atunci cand vom desena o scena si framebufferel va fi legat la banda grafica procesul de desenare va scrie rezultatele (pixelii) in framebufferul legat de noi. Acelasi proces se intampla si pentru framebufferul default dar pana acum nu ne-am lovit de acest detaliu.

Pentru a atasa o textura cu format de culoare (R, RG, RGB, RGBA) deja creata la un framebuffer folosim comanda:

```
glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0+pct_atasare, textura,0);
```

Astfel atasam textura „textura” la framebuffer-ul actual legat la banda grafica pe punctul de atasare pct_atasare. 0-ul de la final ne spune ca atasam primul nivel din mipmap (rezolutia maxima). Dupa cum se poate observa framebufferele au puncte de atasare, foarte similare ca si concept cu pipe-urile folosite la trimiterea atributelor de vertecsi. In OpenGL acest tip de design este foarte des folosit. Daca atasam o textura la un punct de legare pe care deja este legata o alta textura, legatura veche se va pierde si va ramane doar cea noua.

Mai putem observa ca punctul de atasare este de tip GL_COLOR_ATTACHMENT. Mai exista alt tip de punct de atasare, cu un singur punct (unic!) folosit pentru bufferul de adancime, numit GL_DEPTH_ATTACHEMENT. Pentru a lega o textura de adancime (cu format intern GL_DEPTH_COMPONENT) putem folosi comanda:

```
glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, textura_adancime,0);
```

Motivul pentru care am dori sa legam o textura de adancime la un framebuffer este urmatorul: daca nu am avea un buffer de adancime atunci cum s-ar putea realiza testul de adancime stiind ca el are nevoie de un spatiu de stocare pentru adancimea de pe fiecare pixel.

Pentru a putea folosi framebufferul mai sunt inca doua etapa de efectuat: setarea bufferelor de desenare si verificarea statusului framebufferului.

Pentru a seta bufferele de desenare folosim:

```
std::vector<GLenum> drawbuffers;
drawbuffers.push_back(GL_COLOR_ATTACHMENT0+attachment_index_color_texture);
glDrawBuffers(drawbuffers.size(),&drawbuffers[0]);
```

Pratic, cu comanda glDrawBuffers setam care sunt bufferele in care OpenGL deseneaza. In exemplul de mai sus avem o singura textura atasata pe atasamentul de culoare cu numarul 0, care o adaugam intr-un vector pe pozitia 0. Daca avem framebufferul in cauza legat la banda grafica si in fragment shaderul executat avem:

```
layout(location = 0) out vec4 out_color;
```

atunci orice este scris in out_color va fi scris in textura. Evident, acest mecanism poate functiona cu mai multe texturi, ca de exemplu:

```
layout(location = 0) out vec4 out_color;
layout(location = 1) out vec3 color2;
layout(location = 2) out int some_int_buffer;
layout(location = 3) out float some_float_buffer;
```

Unde pe atasamentul de culoare numarul 0 merge ce e scris in out_color, pe atasamentul de culoare cu numarul 1 merge ce este scris in color2, pe atasamentul de culoare cu numarul 2 merge ce este scris in some_int_buffer iar pe atasamentul de culoare cu numarul 3 merge ce este scris in some_float_buffer. Dupa cum se poate observa bufferele pot avea tipuri de date diferite.

Ultima etapa necesara inainte de folosirea framebufferului este testarea corectitudinii sale cu comanda:

```
glCheckFramebufferStatus(GL_FRAMEBUFFER);
```

Pentru a lega framebufferul construit trebuie sa folosim

```
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer_object);
```

iar pentru a lega framebufferul default putem folosi:

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Render to texture

Pentru a desena intr-una sau mai multe texturi folosim urmatorul proces:

- Construim un framebuffer in care sunt legate toate texturile in care dorim sa scriem
- Legam acest framebuffer la banda grafica
- Facem `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);` care va reinitializa valorile din bufferele din framebufferul cu care lucram (pentru ca este legat la banda grafica in acest moment).
- Desenam scena in mod normal, doar ca in fragment shader avem grija ca outputul sa mearga exact pe punctele de atasament specificate in framebuffer (dintr-o decizie de design de cod shaderul va fi identic, dar locatia 0 va avea sensuri diferite: 0 in framebufferul default si 0 in framebufferul creat de noi)
- Dezlegam framebufferul de la banda grafica
- Facem `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);` care va reinitializa valorile din bufferele din framebufferul **default** (legat acum la banda grafica)
- Legam texturile de la framebufferul pe care l-am construit la banda grafica (ele raman legate si la framebuffer)
- Acum le putem mapa texturile precedente la orice obiect cu coordinate de texturare.

Reflectie

Reflectia este fenomen care nu este reprezentabil corect in banda de rasterizare fara a utiliza metode extrem de complicate (este inca o problema **open**). Din acest motiv nu ne propunem corectitudine ci doar sa simulam acest fenomen.

Pentru aceasta consideram ca oglinda functioneaza la randul ei ca o camera si consideram ca imaginea generata de aceasta camera este apropiata de reflectia pe care am observa-o daca oglinda ar functiona corect dpdv fizic.

Algoritm (folosind metoda Render to texture):

1. Se creeaza un framebuffer care contine o textura de culoare direct mapabila pe geometria de suport a oglinzii.
2. Se leaga framebufferul creat la punctul 1 la banda grafica.
3. Se deseneaza scena (fara oglinda) din perspectiva oglinzii. Rezultatul se salveaza in framebufferul creat la punctul 1 (legat la banda grafica in acest moment).
4. Se dezleaga framebufferul. Acum framebufferul default este cel legat la banda grafica

5. Se deseneaza scena din perspectiva camerei din scena. Scena include oglinda.
6. Peste oglinda se mapeaza textura de culoare din framebufferul de la pct 1.

Rezultatul vizual al laboraturlui in varianta nerezolvata:

Rezultatul vizual al laboratorului in varianta rezolvata: