SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1912

PRESLIKAVANJE TEKSTURE STABLOM TEKSTURNIH PLOČICA

Nikola Martinec

Zagreb, lipanj 2011.

Sadržaj

Uvod 1

2. Oktalno stablo 1

3. Kreiranje oktalnog stabla 2

   3.1 Osnovni pristup 2

   3.2 Implementacija oktalnog stabla 5

      3.2.1 Statička izgradnja stabla 5

      3.2.2 Dinamička izgradnja stabla 8

      3.2.3 Usporedba algoritama 9

4. Aplikacija za bojanje objekta 12

5. Volumne teksture 16

Zaključak 19

7. Literatura 20

Sažetak 21

Abstract 21

Privitak 22

Uvod

Modeliranje neke scene samo pomoću osnovnih grafičkih primitiva, npr. poligona, može postati vrlo neučinkovito. Ako bi željeli modelirati zid načinjen ciglama morali bi imati više tisuća poligona koji bi predstavljali pojedine cigle. Konačan rezultat bio bi vizualno neatraktivan, a i korištenje takvog modela negativno bi utjecalo na performanse aplikacije u kojoj se koristi. Korištenje tekstura rješava te probleme. Preslikavanje teksture je, najjednostavnije rečeno, dodjeljivanje slike poligonu. Tako se problem modeliranja zida jednostavno rješava preslikavanjem teksture zida na jedan ili više poligona. Izvor za teksturu može biti računalno izrađena slika, ali i npr. skenirana fotografija.

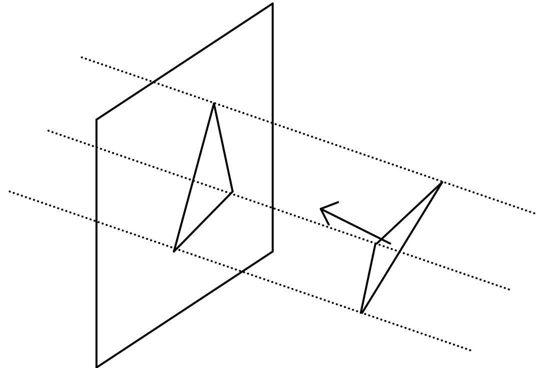
Preslikavanje tekstura je jedna od osnovnih sastavnica rada s računalnom grafikom. Osnovni pristup sastoji se od preslikavanja pravokutnih tekstura na površinu objekta. Kako bi to bilo moguće potrebno je parametrizirati površinu u 2D koordinatni sustav. Vrlo često taj zadatak nije lagan te se javljaju pogreške poput istezanja ili skupljanja konačne teksture na objektu. Zbog tih problema razvijen je novi pristup preslikavanja tekstura. Osnovni cilj je bio izbjeći parametrizaciju površine objekta te se tako rodila ideja o preslikavanju teksture pomoću oktalnog stabla.

2. Oktalno stablo

Oktalno stablo je struktura podataka slična binarnom stablu. Razlika je u tome što svaki čvor oktalnog stabla ima 8 čvorova djece ili listova. Klasično preslikavanje tekstura, u kojem se površini pridjeljuju koordinate teksture, postaje izrazito komplicirano kada se radi o površinama koje je teško parametrizirati kao što su implicitne krivulje. Ideja je napraviti paralelnu projekciju malih dijelova površine na kvadrate te tako preslikati teksturu.

Npr. jedan mali dio površine okružen kockom paralelnom projekcijom projicirat će se na stranicu kocke prema kojoj je usmjeren njegov vektor normale (slika 1.1).

Jedna kocka tako može sadržavati 6 tekstura. Pritom je potrebno paziti da je preslikavanje injektivno, odnosno da ne postoje dvije površine koje se preslikavaju na istu stranicu kocke. Kako bi se to spriječilo potrebno je napraviti dovoljno male kocke koje će zauzimati dovoljno mali dio površine.



Slika 1.1. Preslikavanje teksture

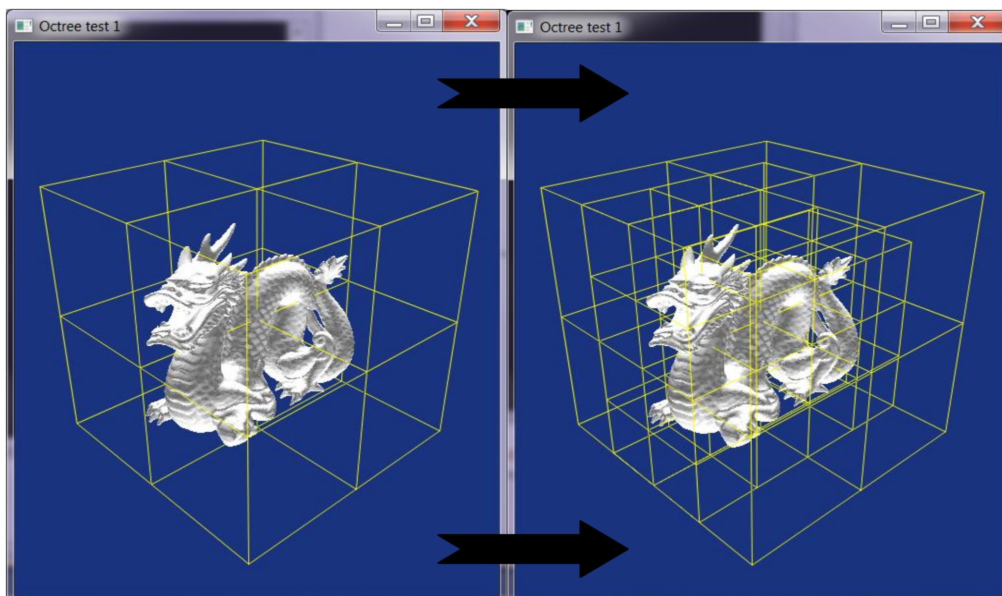
U ovom radu razrađena je samo jednostavnija verzija preslikavanja teksture u kojoj se kao teksturna pločica uzima samo boja određenih poligona. Zbog toga tijekom bojanja nije moguće bojati unutar samog poligona, nego samo po poligonu kao cjelini ili po skupini poligona.

3. Kreiranje oktalnog stabla

3.1 Osnovni pristup

Postupak započinje kreiranjem kocke oko objekta za kojeg želimo napraviti teksturu. Početna kocka mora obuhvaćati cijeli objekt te je ona ujedno i korijen oktalnog stabla. Vrlo je korisno kocku pozicionirati tako da su stranice kocke poravnate s koordinatnim osima što uvelike olakšava neke provjere koje će biti kasnije potrebne. Nakon što smo kreirali kocku odgovarajuće veličine dijelimo ju na

osam manjih, jednakih po veličini, kocaka. Za svaku novo nastalu kocku provjeravamo da li sadrži neke poligone ili je prazna. Provjeru da li kocka sadrži poligon reducirat ćemo na provjeru da li kocka sadrži točku tog poligona. Točka na poligonu je jednostavno odabrana kao težište trokuta. Ako kocka sadrži barem jednu takvu točku dijelimo ju dalje, inače ju odbacujemo. Slika 2.1 prikazuje izgradnju stabla na dubini 2. Sa lijeve strane vidljiva je podjela osnovne kocke na 8 manjih pošto svaka od njih sadrži poligone. Na lijevoj strani je oktalno stablo na dubini 2 koje ima i odbačene kocke koje ne sadrže poligone. Što je dublje oktalno stablo to se više kocke okupljaju oko površine objekta. Postupak je rekurzivan i zaustavlja se na unaprijed zadanoj dubini.



Slika 2.1 Prijelaz oktalnog stabla s dubine 1 na dubinu 2

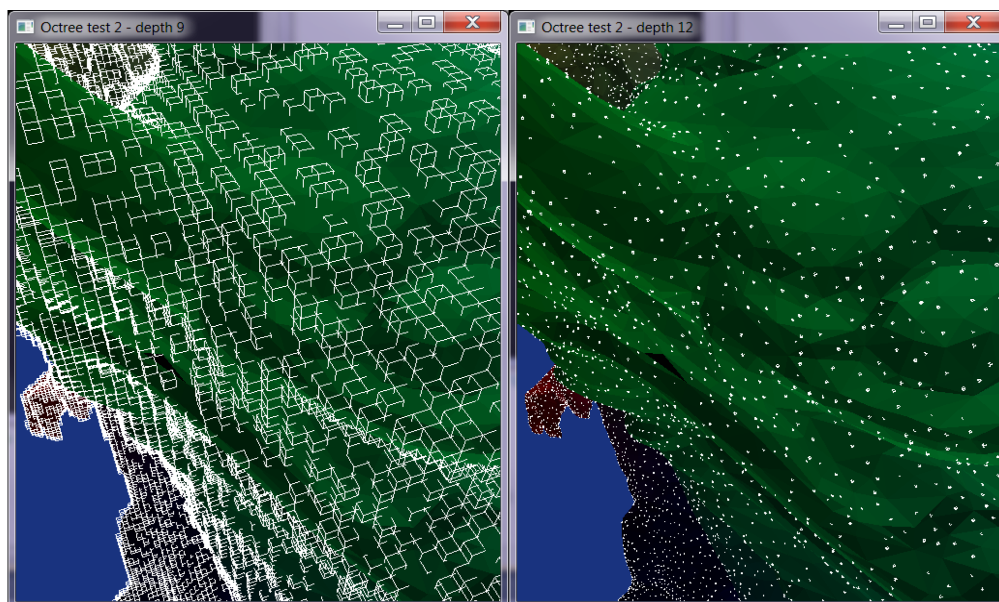
Na određenoj većoj dubinu svaki poligon će pripadati jednoj kocki te u tom trenutku više nema potrebe za dubljim stablom. Daljnjim povećavanjem dubine samo se smanjuje kocka koja obuhvaća točku unutar poligona te se povećava potrošnja memorije zbog referenci s roditeljskog čvora na dijete. Slika 2.2 prikazuje takav slučaj. Da bi se bolje vidjeli listovi stabla kocke ostalih čvorova na toj slici nisu iscrtane. Zbog navedenoga, nema potrebe povećavati dubinu kada broj listova dosegne broj poligona.

U čvorovima na najdubljoj razini, odnosno u listovima, biti će pohranjena boja za sve

poligona koji se nalaze u tom listu. Time će biti stvorena jednostavna tekstura koja će biti namijenjena samo određenom broju poligona.

Ovakav algoritam je poprilično spor jer je za složene objekte koji se sastoje od mnoštva poligona potrebno obaviti mnogo ispitivanja. Kako bi provjerili da li određena kocka sadrži neki poligon ili je prazna potrebno je, u najgorem slučaju, proći kroz sve poligone objekta. Npr., ako bi bila zadana dubina 1 to bi značilo da moramo jednom proći kroz poligone objekta na dubini 0 i još osam puta na konačnoj dubinu 1. Na većim dubinama, brzina izvođenja takvog programa bitno opada. Za dubinu 8 i broj poligona 20 000 broj provjera se počinje mjeriti u milijardama. Zbog toga je bilo potrebno promijeniti pristup.

Umjesto kreiranja oktalnog stabla provjeravajući da li kocka na određenoj dubini sadrži poligone, za svaki poligon odredit ćemo list kojem pripada. Time smo uvelike smanjili broj potrebnih provjera kod izrade stabla te se vrijeme izvršavanja programa bitno ubrzali.



Slika 2.2 Nepotrebno povećavanje dubine podjele oktalnog stabla. Na slici lijevo je dubina 9, a desno 12.

3.2 Implementacija oktalnog stabla

3.2.1 Statička izgradnja stabla

Obje implementacije izvedene su pomoću rekurzije. Zbog toga što je druga (brža) verzija nešto kompliciranija prvo ćemo razmotriti prvu.

Koordinate objekta transformirane su tako da su maksimalne vrijednost x, y i z koordinata 0.5, a minimalne -0.5;

Svaki čvor stabla sastoji se od varijabli *color* u koju se sprema boja ako se radi o listu, *box* u kojoj su spremljene dimenzije kocke koja pripada tom čvoru, *depth* koja pamti dubinu čvora i liste *children* u kojoj su spremljene reference na objekte koji su djeca trenutnog čvora. Algoritam je prikazan u nastavku.

```
private void createNode(Octree_v1 node)
        {
            if (node.depth == paintingDepth)
            {
                node.color.R = node.box.topLeftNearCorner.x;
                node.color.G = node.box.topLeftNearCorner.y;
                node.color.B = node.box.topLeftNearCorner.z
                return;
            }
            for (int i=1; i<=8; i++)
            {
                Box childBox;
                childBox = calculateSubBox(i,node.box);
                if (intersectsPolygons(childBox, node.depth))
                {
                    Octree_v1 child
                    child = new Octree_v1(node.depth + 1,childBox)
                    node.children.Add(child)
                    allBoxes.Add(childBox)
                    createNode(child);
                }
            }
```

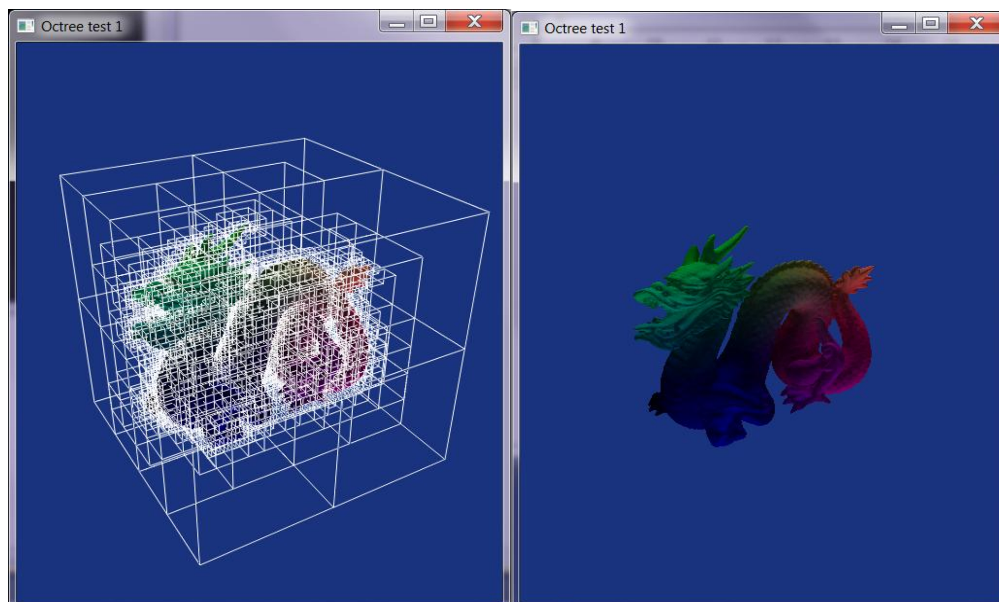
Kreiranje stabla započinje stvaranjem objekta tipa Octree_v1 koji je ujedno i korijen stabla. Konstruktor klase Octree_v1 poziva metodu *createNode* gdje započinje algoritam.

Crveno je označen kod koji služi za izlazak iz rekurzije. Kada se dođe do zadane

dubine postavlja se boja lista na način da se RGB komponentama boje pridruže x, y i z koordinate vrhova kocke (varijable *box*). Od ovakvog načina pridruživanja boje nema previše koristi te on služi samo za provjeru ispravnosti oktalnog stabla. Ovaj dio će biti izbačen kada bude napisan kod za bojanje mreže poligona objekta.

Svaki čvor može imati do 8 čvorova djece, zato imamo petlju koja se izvršava 8 puta. Ovisno o varijabli *i* pozivom metode *calculateSubBox()* generira se kocka u 3D prostoru s odgovarajućim koordinatama. Varijabla *i* određuje poziciju kocke djeteta unutar kocke roditelja. Same dimenzije kocke djeteta su određene kockom roditeljem te je stranica kocke djeteta pola stranice kocke roditelja. Kada imamo koordinate jednog vrha nove kocke vrlo jednostavno možemo provjeriti da li je neka točka u prostoru unutar te kocke. Naime, varijabla *childBox* je zapravo struktura koja se sastoji od koordinate jednog vrha kocke i širine njezine stranice. Pošto je i novo nastala kocka poravnata s koordinatnim osima algoritam za provjeru je vrlo jednostavan i obavlja se pozivom metode *intersectsPolygons().* Metoda prolazi kroz sve poligone objekta dok ne nađe poligon koji je unutar kocke. Ako ga nađe prekida s daljnjim provjerama za ostale poligone i vraća vrijednost *true*. Ukoliko ne uspije naći takav poligon vraća vrijednost *false*.

Ako je metoda *intersectsPolygons* vratila vrijednost *true* ulazi se u programski odsječak označen sivom bojom. Ulazak u njega znači da smo pronašli dijete roditeljskog čvora koji nije prazan (njegova kocka sadrži barem jedan poligon). Kocku spremamo u listu *allBoxes* kako bi kasnije mogli vizualno rekonstruirati oktalno stablo. Zatim instanciramo novi čvor kojemu kao parametre konstruktora predajemo dubinu uvećanu za jedan te samu kocku. Referencu na taj čvor spremamo u listu djece (*node.children*) trenutnog čvora. Rekurzivni postupak se nastavlja u dubinu te stoga ponovno pozivamo metodu *createNode(),* ali ovaj put kao parametar šaljemo referencu na stvoreno dijete *child*. Rekurzija završava kada se postigne željena dubina (označeno crveno) te se listu stabla pridjeljuje boja na već ranije opisan način.



Slika 3.1 Rezultat primjene teksture iz stabla

Za potrebe testiranja korišten je model zmaja koji se sastoji od 50 000 poligona. S lijeve strane slike 3.1 iscrtan je zmaj s pripadnim oktalnim stablom na dubini 5. Može se jasno vidjeti da mnogi roditeljski čvorovi nemaju svu svoju djecu. Prazni čvorovi se zanemaruju te njihove kocke nisu niti iscrtane. Najgušće su raspoređene kocke pri samoj površini objekta i to na maksimalnoj dubini.

S desne strane nalazi se zmaj iscrtan koristeći, pripadno mu, oktalno stablo. Boja pojedinih poligona je definirana tijekom izrade oktalnog stabla i, kako smo već rekli, ovisi o poziciji kocke kojoj površina zmaja pripada. Na taj način dobiveno je prelijevanje boja od zelene, plave prema crvenoj.

Na testnom računalu [1] vrijeme potrebno za stvaranje oktalnog stabla ovim algoritmom na dubini 5 trajalo je 1s, za dubinu 6 6s, a već za dubinu 7 dugih 25s! Zbog loših performansi moramo primijeniti drugačiji pristup. Slijedi drugi, brži algoritam.

1 AMD Phenom II 945 3,0GHz, 4GB RAM

3.2.2 Dinamička izgradnja stabla

```
public Color4 SearchPoint(MyPoint3D point, Octree_v1 node)
        {
            if (node.depth + 1 == paintingDepth)
            {
                node.color.R = node.box.topLeftNearCorner.x;
                node.color.G = node.box.topLeftNearCorner.y;
                node.color.B = node.box.topLeftNearCorner.z;
                return node.color;
            }
            bool isNewChild;
            for (int i=1; i<=8; i++){
                Box childBox;
                Octree_v1 oldChild = null;
                isNewChild = true;
                childBox = calculateSubBox(i,node.box);
                if (pointInBox(point, childBox)){
                    foreach(Octree_v1 oneChild in node.children)
                    {
                           if(oneChild.box.topLeftNearCorner.Equals
(childBox.topLeftNearCorner))

                        {
                            isNewChild = false;
                            oldChild = oneChild;
                        }
                    }
                    if (isNewChild == false)
                    {
                        return SearchPoint(point,oldChild);
                    }
                    else
                    {
                        Octree_v1 child;
                        child = new Octree_v1(node.depth + 1,childBox);
                        node.children.Add(child);
                        Octree_v1.allBoxes.Add(node.box)
                        return SearchPoint(point,child);
                    }

                }

            }
            return new Color4(1,1,1,1);

        }
```

Ideja ovog algoritma je postepeno stvarati oktalno stablo prilikom iscrtavanja poligona objekta. Unutar bloka *GL.Begin(BeginMode.Triangles) – GL.End()* za svaki poligon (za koje smo već rekli da su trokuti) određuje se boja i to metodom *GL.Color4(tree.SearchPoint(point, tree)).* Za svaki poligon poziva se navedena

metoda s parametrima *point* – točka unutar poligona i *tree* – referenca na objekt *tree* koji predstavlja korijen stabla. Na taj se način za svaki poligon, prilikom iscrtavanja, prođe kroz stablo i istovremeno stvore čvorovi na putu do lista kojemu pripada poligon. Samo iz ovoga je vidljiva značajna ušteda vremena potrebnog za stvaranje stabla – više nije potrebno za svaki čvor stabla provjeravati da li sadrži barem jedan poligon, nego samo da li sadrži zadani poligon (predstavljen jednom točkom)!

Osnovna razlika u odnosu na stari algoritam je ta što uvodimo pojam starog djeteta (*oldChild*). Prilikom prvog prolaska stablom za jedan poligon instanciramo čvorove koji se nalaze na putu od korijenskog čvora do lista u kojem se nalazi poligon. Kada drugi put, za sljedeći poligon, prolazimo kroz stablo može se dogoditi da se dio puta od korijenskog čvora do lista preklapa s nekim od prijašnjih putova. Zbog toga smo uveli varijablu *oldChild.* Svaki put kada metodom *pointInBox()* utvrdimo da poligon pripada kocki moramo provjeriti da li već postoji dijete trenutnog čvora u listi djece (*node.children)* s istom kockom (varijablom *box)!* Ta provjera se obavlja u djelu koda označenim zelenom bojom. Ako nađemo takvo dijete ne trebamo stvarati novi čvor već iskoristiti postojeći te pozvati rekurzivno metodu SearchPoint() s parametrima *point* i *oldChild. OldChild* je referenca na čvor dijete koji smo u nekom prijašnjem prolasku kroz stablo instancirali i dodali u listu djece. Drugi slučaj je kada ne postoji dijete trenutnog čvora s kockom u kojoj se nalazi poligon (predstavljen varijablom point). Tada ulazimo u dio koda označen sivom bojom. Taj dio je gotovo jednak sivo označenom djelu prošlog algoritma. Jedina razlika je u pozivu rekurzivne metode.

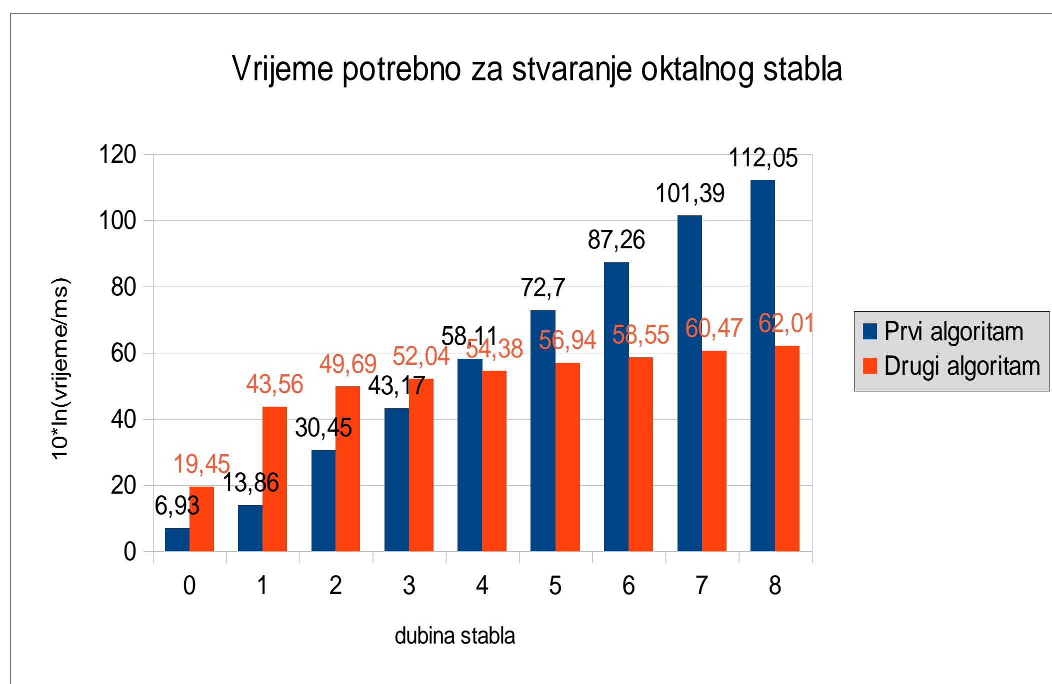
Metoda *SearchPoint()* se poziva za svaki poligon posebno te zbog toga ima i parametar *point,* dok se kod metode *createNode()* odjednom kreira cijelo stablo te se za svaki čvor (kocku) provjerava da li sadrži poligon o čemu brine metoda *intersectsPolygons().*

3.2.3 Usporedba algoritama

Ubrzanje programa je golemo u odnosu na prvu implementaciju. Za 50 000 poligona i dubinu 5 vrijeme stvaranja oktalnog stabla iznosi 250ms (1s u prvoj verziji), a za

dubinu 7 430ms (25s u prvoj verziji) što je ubrzanje za više od 50 puta! Razlog zbog čega je drugi algoritam toliko brži je u pristupu poligona u listi. Sam algoritam provjere da li je poligon unutar kocke je izrazito brz, ono što ga usporava je iteriranje po listi poligona. Prvi algoritam je spor jer broj dohvaćanja poligona eksponencijalno ovisi o dubini stabla, dok je u drugom konstantan i jednak je broju poligona. Ono što usporava drugi algoritam, povećanjem dubine, je broj provjera da li je poligon u kocki, no taj broj raste linearno.

Prikazana je usporedba jednog i drugog algoritma. Vrijeme potrebno za stvaranje stabla prikazano je na y osi i to u logaritamskoj skali, a na x osi nalazi se dubina stabla gdje je stablo dubine 0 stablo sa samo korijenskim čvorom. Podaci su dobiveni za model zmaja koji je već ranije prikazan.



Slika 3.2 Usporedba prvog i drugog algoritma

Iz priloženog grafa jasno se vidi prednost drugog algoritma. Dok kod prvog algoritma vrijeme raste eksponencijalno kod drugog raste linearno. Zbog toga se prvi algoritam počinje brzo usporavati povećanjem dubine stabla te postaje beskoristan. U daljnjem radu biti će korišten drugi algoritam zbog puno boljih performansi.

Potrebno je i omogućiti interaktivno mijenjanje dubine oktalnog stabla kako bi se moglo bojati po raznim dubinama. Time efektivno mijenjamo veličinu površine objekta koja se boja odjednom. Da bi to omogućili potrebne su neke preinake u programskom kodu za kreiranje oktalnog stabla. Bitno je napomenuti da, iako imaju različita imena, metode c*reateNode, SearchPoint* te sljedeća metoda createTree imaju istu funkciju – kreirati oktalno stablo. Imena su im različite zbog načina na koji rade

(*createNode()* i *SearchPoint())* ili zbog konteksta u kojima se pozivaju (*createTree()*). Pozivom metode *createTree()* proširuje se oktalno stablo. U nastavku je dio koda metode *createTree()* sličan metodi *SearchPoint()*.

```
public void createTree(MyPoint3D point,Octree_v1 node,Color4 parentColor)
     {
         if (node.depth > prevMaxDepth)
         {
             node.color.R = parentColor.R;
             node.color.G = parentColor.G;
             node.color.B = parentColor.B;
         }
         if (node.depth == paintingDepth)
         {
             return;
         }
         bool isNewChild = true;
         for (int i = 1; i <= 8; i++)
         {
             Box childBox;
             Octree_v1 oldChild = null;
             isNewChild = true;
             childBox = calculateSubBox(i,node.box);
        ...
```

Jedina velika razlika je u tome što se sada kao parametar metodi predaje i boja roditeljskog čvora. Zbog toga što se ova metoda poziva i kako bi se stablo proširilo na veću dubinu (npr. sa dubine 2 na dubinu 8) potrebno je propagirati boju s trenutnih listova, koji to odjednom više nisu, njihovoj djeci te konačno na nove listove. Logika je jednostavna – ako smo u čvoru koji je na većoj dubini nego što je bila prijašnja maksimalna dubina (*prevMaxDepth)* moramo naslijediti boju roditeljskog čvora. Ova metoda se poziva za svaki poligon objekta te će se na taj

način rekurzivno propagirati boja s prijašnjih listova na nove. Bitno je napomenuti da se ova metoda poziva kod stvaranja početnog stabla i to dubine 0 te svaki put kada dubinu povećamo te je ona maksimalna (sve prijašnje odabrane dubine su manje). Ostatak koda je gotovo identičan kodu metode *SearchPoint()* te ga nema potrebe ponovno ponavljati.

4. Aplikacija za bojanje objekta

Korištenje vrhova kocaka u listovima stabla kao izvor podataka za boju koja će biti pohranjena u listu nema neku praktičnu korist osim provjere ispravnosti stabla. Zbog toga je potrebno napisati aplikaciju koja će omogućiti interaktivno bojanje 3D objekta. Odabirom dubine stabla eksplicitno odabiremo rezoluciju na kojoj će se odvijati bojanje. Svi listovi koji pripadaju jednom listu biti će obojeni bojom spremljenom u tom listu.

Kada korisnik klikne na objekt iscrtan u prozoru na ekranu određuje se poligon koji je na tim koordinatama. Zatim se traži njegovo mjestu u oktalnom stablu te se pripadnom listu pridjeljuje unaprijed odabrana boja. Prilikom ponovnog iscrtavanja objekta svi poligoni koji pripadaju tom listu biti će iscrtani jednakom bojom.

Za određivanje poligona koji je odabran iskorišten je jednostavan algoritam. Prilikom učitavanja poligonalne mreže objekta svakom poligonu pridružuje se jedinstvena boja. Broj poligona kojima možemo pridružiti jedinstvenu boju je određen rasponom boja u RGB sustavu i on iznosi više od 16 milijuna (RGB – 256*256*256). Nekome bi to moglo biti premalo pa može iskoristiti i *alpha* komponentu i pomnožiti taj broj još jednom s 256. Ovime smo dobili mogućnost identificiranja poligona preko njegove boje.

Klikom na ekran iscrtava se objekt sa svojim jedinstvenim skupom boja, ali u "back" spremniku. Metodom GL.ReadPixels() dohvaća se boja slikovnog elementa na trenutnoj poziciji miša. Zbog jedinstvenog pridjeljivanja boja odmah možemo iz same boje zaključiti o kojem se poligonu radi. Odabir boje kojom se želi obojiti dio objekta vrši se preko kvadrata u uglu ekrana u kojem su interpolirane crna, crvena,

zelena i plava boja.

Slijedi odsječak koda za postavljanje boje lista u stablu.

```
public void SetColorToLeaf(Color4 newColor,Octree_v1 node,MyPoint3D point,
Color4 parent)
        {
            if (node.depth >= paintingDepth)
            {
                node.color.R = newColor.R;
                node.color.G = newColor.G;
                node.color.B = newColor.B;

                if (node.depth == maxDepth)
                {
                    return;
                }
                foreach (Octree_v1 child in node.children)
                {
                    SetColorToLeaf(newColor,child,point,node.color);
                }
                return;
            }

            foreach (Octree_v1 child in node.children)
            {
                if (PointInBox(point,child.box))
                {
                    SetColorToLeaf(newColor,child,point,node.color);
                    return;
                }
            }
  }
```

Ovaj kod ne izgleda baš intuitivno i potrebno ga je malo više objasniti. Da bi ga mogli shvatiti potrebno je objasniti dva osnovna slučaja koja se mogu dogodi tijekom bojanja objekta.

a) Bojali smo na nižoj dubini i sada želimo bojati na višoj dubini koja je ujedno i maksimalna

Počinjemo u djelu obojanom svijetlo plavo. Cilj je pronaći list kojemu pripada kliknuti poligon te mu postaviti boju koju smo odabrali. Varijabla *paintingDepth* čuva vrijednost dubine na kojoj treba spremiti boju u list. Zbog

toga što smo rekli da smo odabrali novu maksimalnu dubinu, dubina trenutnog čvora nikad neće biti veća od nje. Stoga će se uvjet označen crveno ostvariti baš kad pogodimo list stabla. Nakon toga će list poprimiti novu boju (koja je inicijalno postavljena u bijelu). Odmah poslije toga uvjet u sivom će biti ostvaren jer smo na maksimalnoj dubini te je ulazak u njegovo tijelo ujedno i kraj rekurzije.

b) Ostvario se slučaj pod a) i sada ponovno želimo bojati na nekoj nižoj dubini

Promjena dubine na kojoj želimo bojati ne utječe na stvarnu dubinu stabla jer želimo sačuvati i boju pohranjenu u dubljim listovima. Mijenja se samo dubina na kojoj se boja (*paintingDepth*). Dodatni posao koji se mora obaviti poslije pridjeljivanja boje čvoru na dubini za crtanje je propagiranje te boje na dublje čvorove te konačno listove. Ovo izgleda kao da se kosi s prvom rečenicom, no to nije tako. Naime, globalno zaista želimo sačuvati boju u svim čvorovima, ali lokalno želimo da svi poligoni koji pripadaju čvoru kojem pripada i kliknuti čvor koji se nalazi **na dubini za bojanje** imaju istu boju. Kako bi to ostvarili moramo propagirati boju od čvora na dubini za bojanje (kojemu nakon ulaska u crveni uvjet pridjeljujemo boju) svoj njegovoj djeci, bolje rečeno svim njegovim potomcima i to ostvarujemo rekurzivnim pozivom *setColorToLeaf()* za svako dijete trenutnog čvora. Rekurziju će zaustaviti uvjet u sivoj boji kada dođemo do maksimalne dubine stabla.

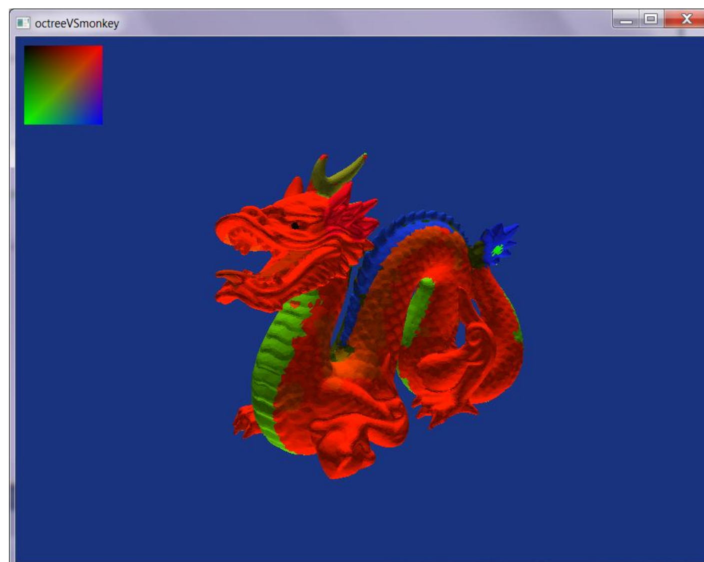
Ostalo je još samo dohvaćanje boje koja se koristi kod iscrtavanja poligona. Ono je vrlo jednostavno ostvareno metodom *GetColorFromLeaf()*. Metoda slijedi u nastavku.

```
public Color4 GetColorFromLeaf(Octree_v1 node,MyPoint3D point)
        {
            if (node.depth == maxDepth)
            {
                return node.color;
            }
            foreach (Octree_v1 child in node.children)
            {

                if (PointInBox(point,child.box))
                {
                    return GetColorFromLeaf(child,point);
                }
            }
            return Color4.White;
        }
```

Metoda se poziva kako bi se odredila boja svakog poligona tijekom iscrtavanja objekta. Radi na način da za dani poligon traži njegov list najdublje moguće u stablu te vraća boju tog lista.

Na slici 4.1 može se vidjeti obojan zmaj pomoću ove aplikacije.



Slika 4.1 Primjer obojanog objekta

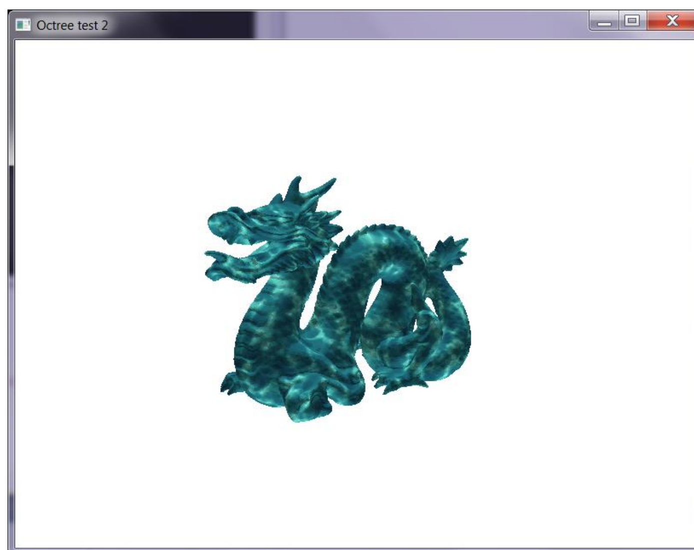
Problem koji se pojavljuje tijekom bojanje objekta je trzanje aplikacije. Trzanje se pojavljuje svaki puta kada se klikne na objekt s namjerom da se oboji određena površina. Razlog trzanja je potreba za učitavanjem kompletne teksture objekta iz oktalnog stabla svaki puta kada u njega, zbog bojanja, unosimo promjene. U aplikaciji su korištene prikazne (engl. Display) liste. One predstavljaju skup OpenGL naredbi koje su pohranjene u memoriji spremne za izvršavanje. Njihova prednost je veliko poboljšanje performansi jer kod jednom definirane lista nema potrebe za ponovnim učitavanjem vrhova i slikovnih elemenata kod svakog iscrtavanja scene. Nedostatak je što je prikazna lista statična. Zbog toga se na svaku promjenu oktalnog stabla mora nanovo kreirati lista, što traje neko vrijeme. Na dubini 9 to vrijeme iznosi oko 80ms što je poprilično osjetno, osobito ako npr. tijekom bojanja rotiramo objekt.

Taj je problem djelomično riješen odgađanjem ponovnog kreiranja prikazna liste te spremanjem odabranih poligona i pripadnih boja u zasebnu, privremenu listu. Ona omogućava glatko bojanje, bez trzanja, po objektu sve dok je tipka na mišu pritisnuta – doduše uz jedno veliko ograničenje. Ako je uključena ova opcija možemo bojati samo po poligonima bez obzira na dubinu! Jednom kada otpustimo tipku miša prikazna lista se ponovno kreira, a memorija za privremenu lista se oslobađa. Tek tada će se iskoristiti podaci o boji iz oktalnog stabla te će objekt biti ispravno obojan.

5. Volumne teksture

Volumne teksture su zapravo trodimenzionalna polja slikovnih elemenata. Česte se preslikavanje volumnih tekstura uspoređuje sa rezbarenjem nekog predmeta iz drva ili kamena (Slika 5.1). Takva analogija je malo "škakljiva". Kada bi volumna tekstura zauzela prostor nad kojim je definirana (npr. 128*128*128 slikovnih elemenata) dobili bi kruti objekt koji bi, ovisno o teksturi, mogao predstavljati npr. drvo ili mramor. Preslikavanje teksture vrši se na način da se za svaki slikovni element objekta određuje njegov položaj unutar teksture te se pripadni slikovni element

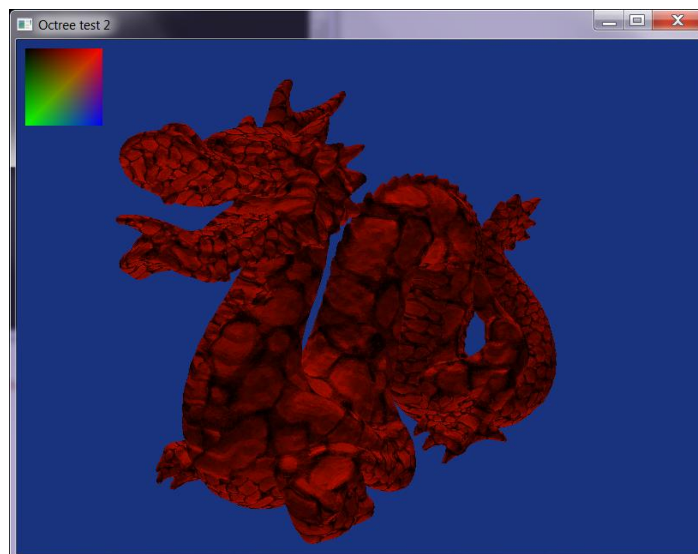
teksture preslikava na element. Najčešće se pri tom koriste interpolacijski postupci pri određivanju slikovnog elementa za preslikavanje. Razlog usporedbe sa rezbarenjem je upravo efekt koji se postiže s volumnim teksturama koje su generirane tako da oponašaju pojedine materijale.



Slika 5.1 Primjer iscrtanog objekta korištenje volumne teksture

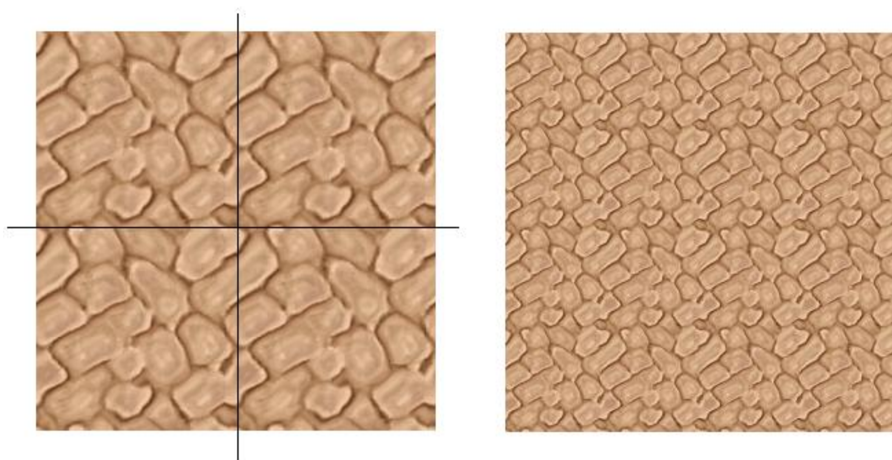
Oktalno stablo moguće je iskoristiti i za raspodjelu volumnih tekstura po objektu. Volumne teksture se lako primjenjuju pošto nema potrebe za parametrizacijom površine. Jedino što je potrebno su vrhovi trokuta na koje se teksture preslikava.

Proširenje programa da radi i s volumnim teksturama nije komplicirano. Uz boju koja se sprema u listovima dodan je novi atribut - gustoća volumne teksture (Slika 5.2, primjer različite gustoće volumne teksture). Gustoća početnog uzorka teksture se lako poveća na način da se poveća raspon koordinata objekta prilikom preslikavanja teksture. Na slici 5.3 vidi se ponavljanje uzorka povećanjem gustoće teksture. Na lijevoj teksturi vidljivo je ponavljanje osnovnog uzorka 4 puta, a na lijevoj 16.



Slika 5.2 Korištenje volumnih tekstura različite gustoće

Npr. neka je neka površina definirana na rasponu [0,1]. Uzmimo da je i tekstura koji želimo preslikati definirana na istom rasponu. Ako raspon koordinata površine pomnožimo s dva dobit ćemo novi raspon koji sada premašuje raspon teksture. Zbog načina na koji openGL radi preslikavanje, za elemente veće od 1 ponovno će se uzimati tekstura od vrijednosti 1 pa na dalje. Mogli bi smo reći da su zbog toga koordinate teksture u nekoj petlji koja rezultira ponavljanjem uzorka. Zbog potrebe definiranja gustoće teksture omogućena su dva načina rada programa. Jedan omogućava promjenu dubine stabla te definiranje boje listova, a drugi umjesto boje definira gustoću teksture.



Slika 5.3 Volumne teksture različite gustoće primijenjena na kvadrat

Zaključak

Ovim radom implementirani su algoritam za stvaranje oktalnog stabla i bojanje objekta. Konačni rezultat je mogućnost bojanja mreže poligona, od koje je sastavljen objekt, na različitim dubinama stabla te korištenje oktalnog stabla kao strukture za spremanje i lako dohvaćanje teksture koja se sastoji od skupa boja. Također, kao dodatak, oktalno stablo iskorišteno je kod preslikavanja volumnih tekstura.

Postoje neka ograničenja ovog rada koja ne dozvoljavaju praktičnu primjenu. Prvo, a i najveće, je činjenica da nemamo mogućnost pohranjivanja više različitih boja unutar samih poligona, već je poligon najniži element bojanja dok bi realno to trebao biti slikovni element. Razlog zbog kojeg to nije ostvareno u ovom radu je nedostatak iskustva zbog kojeg sam krenuo s ovim jednostavnijim pristupom.

Unatoč tome, ovaj rad mi je pomogao u shvaćanju problema koje treba savladati kako bi se napisala aplikacija za "pravo" preslikavanje tekstura koristeći oktalno stablo. Štoviše, ovim radom sam prikupio puno iskustva u radu s OpenGL-om kojeg nisam mogao steći bez bacanja u koštac s raznim problemima ovog rada.

7. Literatura

[1] BENSON, D., AND DAVIS, J. "Octree textures". Proceedings of ACM SIGGRAPH, ACM Press, 2002.

[2] LEFABVRE, S., AND DACHSBACHER, C. "TileTrees", Proceedings of ACM Symposium on Interactive 3D Graphics and games, ACM Press, 2007.

[3] SUTER, J. "Introduction To Octrees", 13. travnja 1999., http://www.flipcode.com/archives/Introduction_To_Octrees.shtml, 20. ožujka 2011.

[4] LEFABVRE, S., HORNUS, S., NEYRET, F. "Octree Textures on the GPU", GPU Gems 2, http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html, 14. travnja 2011.

[5] Tutorials:OpenGL Lighting, http://www.spacesimulator.net/wiki/index.php/Tutorials:OpenGL_Lighting, 5. travnja 2011

[6] OpenGL Selection Using Unique Color IDs, http://www.spacesimulator.net/wiki/index.php/Tutorials:OpenGL_Lighting, 2. svibnja 2011

[7] SONG HO AHN. "OpenGL Display List", 2005., http://www.songho.ca/opengl/gl_displaylist.html, 14. travnja 2011.

Sažetak

Preslikavanje tekstura upotrebom oktalnog stabla je relativno novi pristup. Njegova prednost je korištenje 2D tekstura bez potrebe za parametrizacijom objekta. U ovom radu objašnjen je postupak kreiranja oktalnog stabla i spremanje teksture, doduše samo u obliku boje određenog poligona. Glavna ideja je spremanje podataka o teksturi u listove oktalnog stabla. Objašnjena su dva algoritma za kreiranje oktalnog stabla te je dana usporedba tih algoritama u vidu vremena izvođenja. Za potrebe stvaranje teksture napisana je i aplikacija za bojanje poligona. Dodatno, kreirano oktalno stablo korišteno je i prilikom preslikavanja volumnih tekstura, odnosno raspoređivanja volumnih tekstura.

Puni naslov zadatka rada: Preslikavanje teksture stablom teksturnih pločica

Ključne riječi: oktalno stablo, preslikavanje tekstura

Abstract

Texture mapping using an octree is relatively new approach. The advantage of using it is avoiding mesh parameterization that is usually everything but simple. In this paper, I have explained the procedure for creating an octree and saving a texture, though only in a form of polygon color representation. The main idea is to save texture data into the leaves of an octree. For that purpose I presented two algorithms and comparison of their execution times. Mesh painting application was written for painting the polygons and by that creating a texture. Created octree was also used for volume texture mapping.

Full title of the task: Texture mapping using texture tiles and an octree

Key words: octree, texture mapping

Privitak

Uz programsku potporu, na kompaktnom disku priložene su i detaljne upute za njeno korištenje.