Problem teselacije NURBS ploha

Mlinarić, Ana

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ana Mlinarić

PROBLEM TESELACIJE NURBS PLOHA

Diplomski rad

Voditelj rada:
Prof. dr. sc. Luka Grubišić

Zagreb, veljača, 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem mentoru prof. dr. sc. Luki Grubišić na stručnom vodstvu te na pomoći, strpljenju i savjetima pruženima prilikom pisanja ovog rada.

Nadalje, hvala svim mojim prijateljima i kolegama koji su mi bili veliki oslonac i učinili da studiranje bude jedan od najljepših životnih perioda.

Za kraj, najveće hvala cijeloj mojoj obitelji, a pogotovo mojim roditeljima, na pruženoj nesebičnoj potpori, razumijevanju i ljubavi kroz sve trenutke mog školovanja.

Svima od srca hvala.

Sadržaj

Uvod

Neuniformne racionalne B-splajn (NURBS) plohe imaju primjenu u područjima kao što su CAD (eng. Computer Aided Design), animacija i virtualna stvarnost.
Modeliranje složenijih objekata NURBS plohama omogućuje visoku kvalitetu rezultata s niskim zahtjevima za memoriju.
Proces generiranja dvodimenzionalne ili trodimenzionalne slike iz modela naziva se renderiranje. Predloženo je nekoliko metoda za renderiranje NURBS ploha, no zbog velikog poboljšanja u hardveru za renderiranje trokutima, metoda koja se najčešće koristi je teselacija. Teselacija plohe je postupak stvaranja mreže trokuta koja aproksimira početnu plohu. Cilj rada je usporediti metode teselacije dostupne u literaturi te prikazati algoritam teselacije na primjerima.
U radu su definirani pojmovi potrebni za razumijevanje algoritama teselacije. Detaljno je objašnjen pojam teselacije te su navedeni poznati algoritmi teselacije trokutima. Objašnjena je uniformna, adaptivna i potpuno adaptivna teselacija. Svi algoritmi za teselaciju NURBS ploha generiraju početnu grubu plohu te je dijele na manje dijelove.
Za implementaciju je korišten OpenGL, sučelje za programiranje grafičkih aplikacija.
Objašnjene su faze teselacije u OpenGL-u te je na primjerima proveden postupak teselacije trokutima.

Poglavlje 1

Parametarske plohe

Jedan od osnovnih ciljeva ovog rada je definirati pojam NURBS (eng. Non Uniform Rational B-Spline) plohe. Prije precizne definicije NURBS plohe, potrebno je definirati nekoliko općenitijih pojmova.
Definicije u ovom poglavlju preuzete su iz knjige [8].
Za prikaz svih grafova u ovom poglavlju, kao i za opis teselacije u drugom poglavlju, korišten je programski jezik Python (verzija 3.6.4) i biblioteka NURBS-Python (geomdl) [6]. NURBS-Python je objektno-orijentirana biblioteka za rad s B-splajn i NURSB plohama i krivuljama. Sadrži implementaciju algoritama računanja s navedenim plohama, opcije vizualizacije ploha, kao i mogućnosti generiranja vektora i ploha.

1.1 Bézierove krivulje i plohe

Definicija 1.1.1. *Bézierova krivulja stupnja n definira se na sljedeći način:*

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i \qquad 0 \leq u \leq 1, \tag{1.1}$$

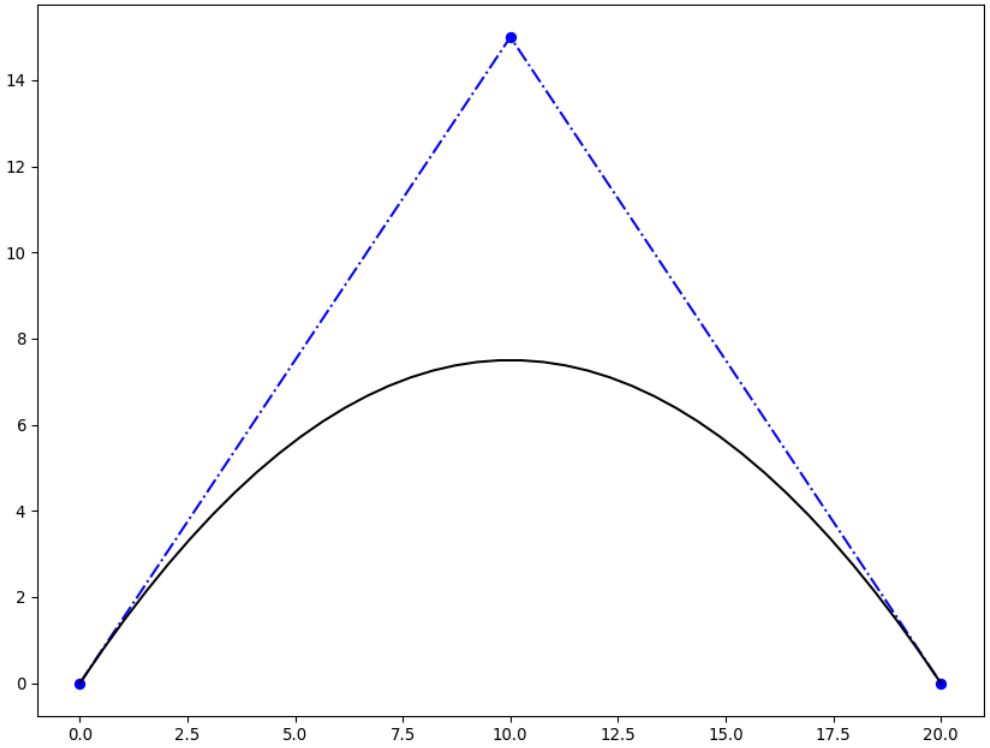
pri čemu se koeficijenti P_i nazivaju kontrolne točke, a funkcije $B_{i,n}$ su Bernsteinovi polinomi dani sljedećom formulom:

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}. \tag{1.2}$$

Primjer 1.1.2. *Neka je stupanj krivulje $n = 2$. Iz jednadžbi* (1.1) *i* (1.2) *slijedi:*

$$\begin{aligned} C(u) &= \sum_{i=0}^{2} \frac{2!}{i!(2-i)!} u^i (1-u)^{2-i} P_i \\ &= \frac{2!}{0!(2-0)!} u^0 (1-u)^2 P_0 + \frac{2!}{1!(2-1)!} u^1 (1-u)^1 P_1 + \frac{2!}{2!(2-2)!} u^2 (1-u)^0 P_2 \\ &= (1-u)^2 P_0 + 2u(1-u) P_1 + u^2 P_2 \end{aligned} \tag{1.3}$$

Tada je $P_0 = C(0)$ *i* $P_1 = C(1)$.
Slika (1.1) *prikazuje graf opisane Bézierove krivulje.*



Slika 1.1: Bézierova krivulja drugog reda

```
from geomdl import BSpline
from geomdl import utilities
from geomdl.visualization import VisMPL

BezierCurve = BSpline.Curve()

BezierCurve.degree = 2
BezierCurve.ctrlpts = [[0, 0], [10, 15], [20, 0]]
BezierCurve.knotvector = utilities.generate_knot_vector(BezierCurve.
    degree, len(BezierCurve.ctrlpts))

BezierCurve.evaluate()
BezierCurve.vis = VisMPL.VisCurve2D()
BezierCurve.render()
```

Isječak koda 1.1: Prikaz Bézierove krivulje

Definicija 1.1.3. *Racionalna Bézierova krivulja stupnja n definirana je sljedećom formulom:*

$$C(u) = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i P_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \qquad 0 \leq u \leq 1 \tag{1.4}$$

Kao i u prethodnoj definiciji, P_i su kontrolne točke, a funkcije $B_{i,n}$ Bernsteinovi polinomi. Koeficijenti w_i se nazivaju težine kontrolnih točaka.

Neka je s $W(u) = \sum i = 0^n B_{i,n}(u) w_i$ označen zajednički nazivnik. Osim ako eksplicitno nije drugačije navedeno, smatra se da je $w_i \geq 0$ za sve i. Ta činjenica osigurava da je $W(u) > 0$ za sve $u \in [0, 1]$. Nadalje,

$$C(u) = \sum_{i=0}^{n} R_{i,n}(u) P_i, \qquad 0 \leq u \leq 1 \tag{1.5}$$

gdje je

$$R_{i,n}(u) = \frac{B_{i,n}(u) w_i}{\sum_{j=0}^{n} B_{j,n}(u) w_j} \qquad 0 \leq u \leq 1 \tag{1.6}$$

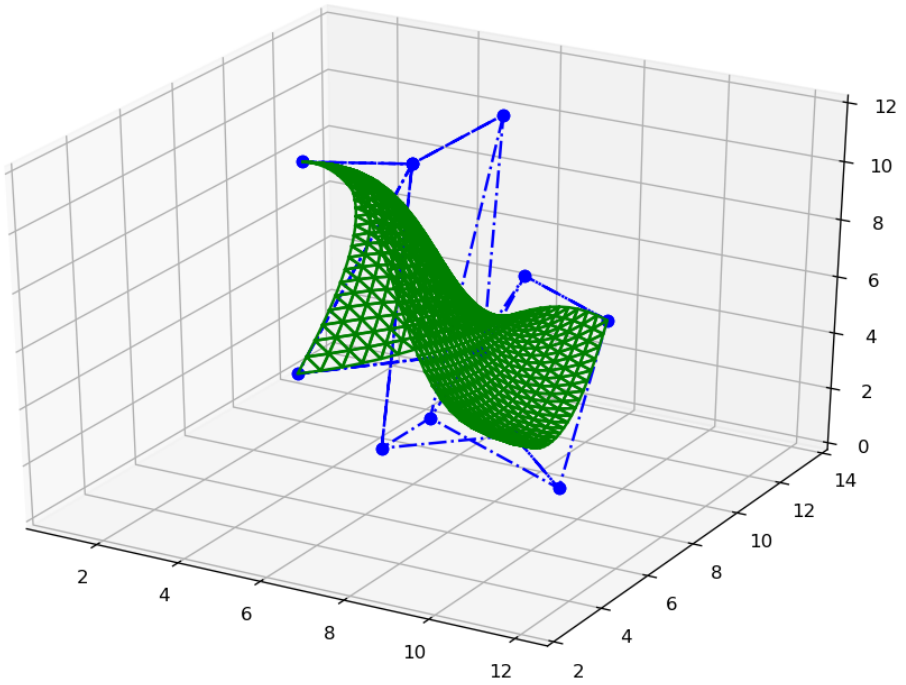
$R_{i,n}$ su racionalne osnovne (bazne) funkcije za opisanu formu krivulje.

Definicija 1.1.4. *Bézierova ploha je definirana sljedećom formulom:*

$$S(u, v) = \frac{\sum_{i=0}^{p} \sum_{j=0}^{q} B_{i,p}(u) B_{j,q}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{p} \sum_{j=0}^{q} B_{i,p}(u) B_{j,q}(v) w_{i,j}}, \tag{1.7}$$

pri čemu su $B_{i,p}(u)$ i $B_{j,q}$ Bernsteinovi polinomi stupnjeva p i q takvi da vrijedi $0 \leq u, v \leq 1$, $P_{i,j}$ su kontrolne točke, a $w_{i,j}$ težine pripadnih kontrolnih točaka.

Primjer 1.1.5. *U ovom primjeru prikazan je graf Bézierove plohe koja je zadana kontrolnim točkama i vektorima čvora prikazanima u isječku koda 1.2.*



Slika 1.2: Bézierova ploha

```
from geomdl import BSpline
from geomdl import utilities
from geomdl.visualization import VisMPL

BezierSurface = BSpline.Surface()

BezierSurface.degree_u = 3
BezierSurface.degree_v = 2

control_points = [[4, 7, 11], [6, 8, 11], [5, 5, 5],
                  [6, 8 , 11], [7, 10, 12], [6, 11, 3],
                  [7, 5, 3], [7, 7, 3], [7, 11, 6],
                  [9, 7, 3], [9, 9, 0], [9, 11, 5]]

BezierSurface.set_ctrlpts(control_points, 4, 3)
BezierSurface.knotvector_u = utilities.generate_knot_vector(
    BezierSurface.degree_u, BezierSurface.ctrlpts_size_u)
```

```
17 BezierSurface.knotvector_v = utilities.generate_knot_vector(
       BezierSurface.degree_v, BezierSurface.ctrlpts_size_v)
18
19 BezierSurface.sample_size = 25
20
21 BezierSurface.evaluate()
22
23 BezierSurface.vis = VisMPL.VisSurface()
24 BezierSurface.render()
```

Isječak koda 1.2: Prikaz Bézierove plohe

1.2 B-splajn krivulje i plohe

Definicija 1.2.1. *Neka je* $U = \{u_0, \ldots, u_m\}$ *neopadajući niz realnih brojeva, takvih da vrijedi* $u_i \leq u_{i+1}, \quad i = 0, \ldots, m-1$. *Realni brojevi* u_i *se nazivaju čvorovi, a* U *vektor čvora. i-ta B-splajn bazna funkcija stupnja* p *(reda* $p+1$*), u oznaci* $N_{i,p}$*, definira se kao*

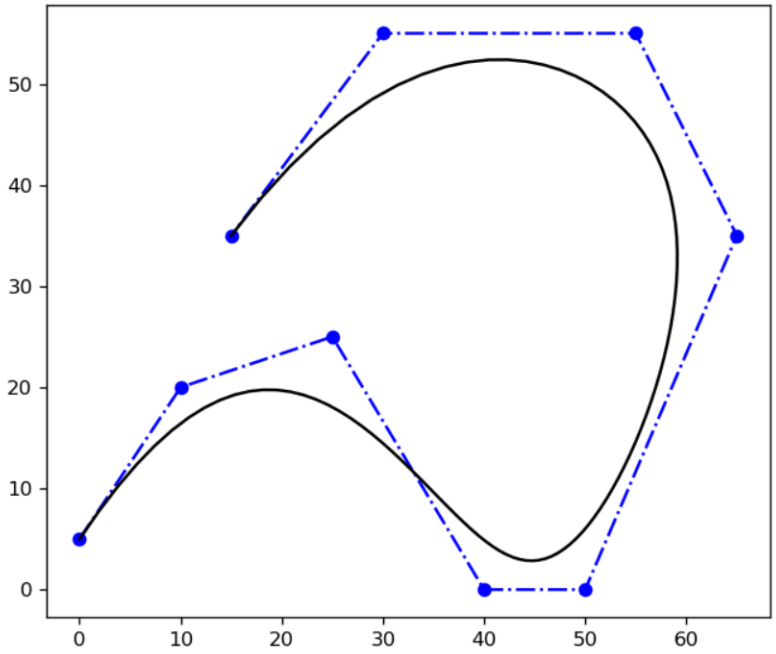
$$N_{i,0}(u) = \begin{cases} 1 & \text{ako } u_i \leq u \leq u_{i+1} \\ 0 & \text{inače} \end{cases} \tag{1.8}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{1.9}$$

Definicija 1.2.2. *B-splajn krivulja stupnja* p *definirana je formulom*

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \qquad a \leq u \leq b, \tag{1.10}$$

pri čemu su P_i *kontrolne točke,* $N_{i,p}(u)$ *B-splajn bazne funkcije stupnja* p *definirane na neperiodičkom i neuniformnom vektoru čvora* $U = \left\{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\right\}$. *Ako nije navedeno drugačije, smatra se da je* $a = 0$ *i* $b = 1$. *Mnogokut čiji su vrhovi* P_i *naziva se kontrolni mnogokut.*



Slika 1.3: B-splajn krivulja

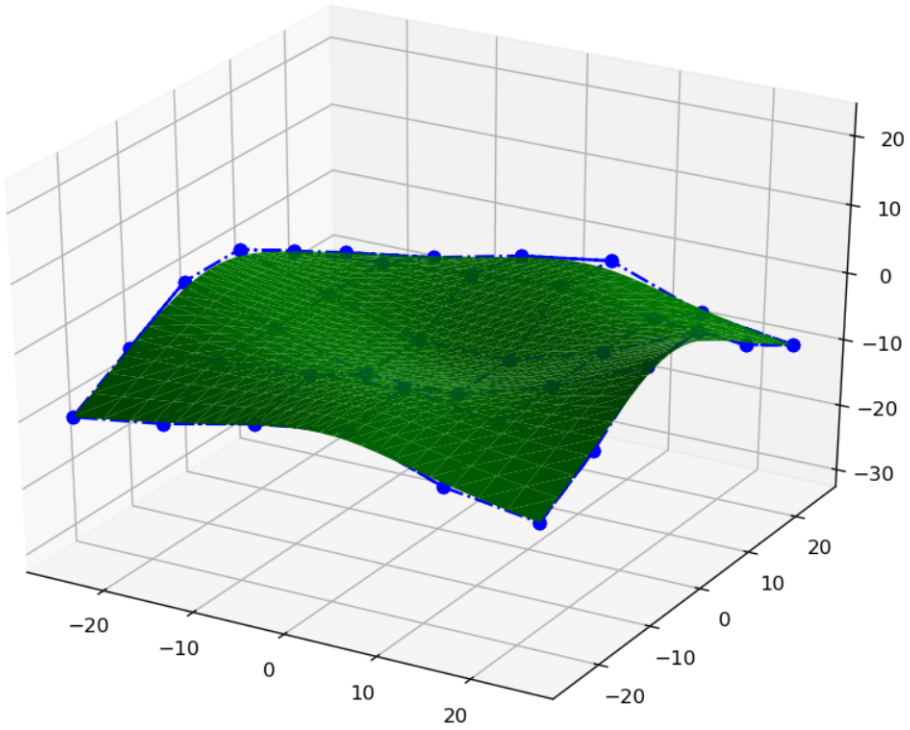
```
from geomdl import BSpline
from geomdl import utilities
from geomdl import exchange
from geomdl.visualization import VisMPL

curve = BSpline.Curve()

curve.degree = 4
curve.ctrlpts = exchange.import_txt("curve.cpt")
curve.knotvector = utilities.generate_knot_vector(curve.degree, len(
    curve.ctrlpts))
curve.delta = 0.01

curve.evaluate()
curve.vis = VisMPL.VisCurve2D()
curve.render()
```

Isječak koda 1.3: Prikaz B-splajn krivulje

Definicija 1.2.3. *B-splajn ploha definirana je sljedećom formulom:*

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p} N_{j,q}(v) P_{i,j}. \tag{1.11}$$

$P_{i,j}$ predstavlja dvosmjernu mrežu kontrolnih točaka, a $N_{i,p}$ i $N_{j,q}$ su B-splajn osnovne funkcije stupnjeva p i q.



Slika 1.4: B-splajn ploha

```
from geomdl import BSpline
from geomdl import exchange
from geomdl.visualization import VisMPL as vis

surf = BSpline.Surface()

surf.degree_u = 3
surf.degree_v = 3
surf.set_ctrlpts(*exchange.import_txt("surface.cpt", two_dimensional=
    True))
```

```
10
11 surf.knotvector_u = [0.0, 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 3.0, 3.0]
12 surf.knotvector_v = [0.0, 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 3.0, 3.0]
13 surf.delta = 0.04
14
15 surf.evaluate()
16 surf.vis = vis.VisSurfTriangle()
17 surf.render()
```

Isječak koda 1.4: Prikaz B-splajn plohe

1.3 NURBS krivulje i plohe

Definicija 1.3.1. *Neuniformna racionalna B-splajn (NURBS) krivulja stupnja p definirana je sljedećom formulom:*

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i} \qquad a \leq u \leq b, \tag{1.12}$$

pri čemu su P_i kontrolne točke koje oblikuju kontrolni mnogokut, w_i težine točaka, a $N_{i,p}(u)$ B-splajn osnovne funkcije stupnja p definirane na neperiodičkom i neuniformnom vektoru čvora $U = \left\{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\right\}$.

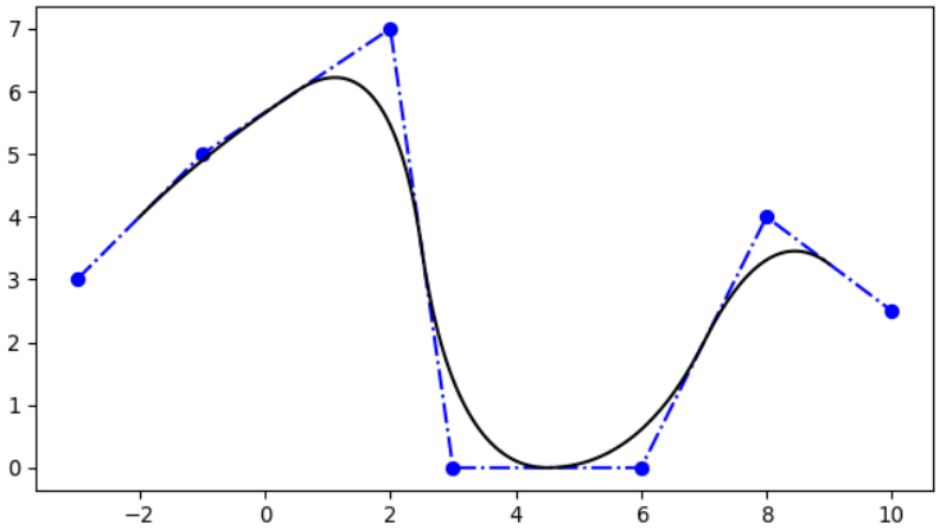
Ako nije drugačije navedeno, pretpostavlja se da je $a = 0$, $b = 1$ i $w_i > 0$ za sve i. Uz oznaku

$$R_{i,p}(u) = \frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j} \tag{1.13}$$

jednadžba (1.13) može biti prikazana u sljedećem obliku:

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u) P_i \tag{1.14}$$

$R_{i,p}(u)$ se nazivaju racionalne bazne funkcije i one su po dijelovima racionalne za $u \in [0, 1]$.
U nastavku je prikazan graf NURBS krivulje.



Slika 1.5: NURBS krivulja

```
from geomdl import NURBS
from geomdl.visualization import VisMPL

curve = NURBS.Curve()

curve.degree = 2
curve.ctrlpts = [[-3, 3], [-1, 5], [2, 7], [3, 0], [6, 0], [8, 4], [10,
    2.5]]
curve.knotvector = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
curve.sample_size = 100

curve.evaluate()
curve.vis = VisMPL.VisCurve2D()
curve.render()
```

Isječak koda 1.5: Prikaz NURBS krivulje

Definicija 1.3.2. *Neuniformna racionalna B-splajn (NURBS) ploha stupnja p definirana je sljedećom formulom:*

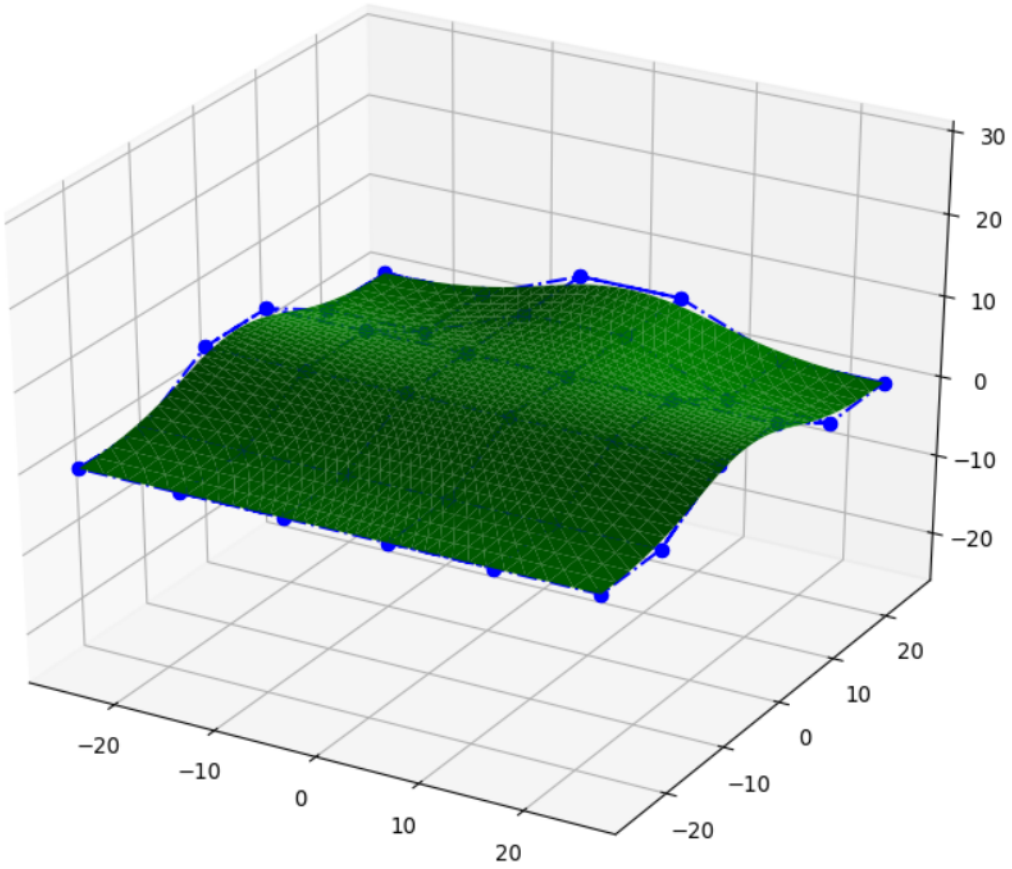
$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \leq u,v \leq 1, \tag{1.15}$$

pri čemu je $P_{i,j}$ dvosmjerna mreža kontrolnih točaka, $w_{i,j}$ su težine, a $N_{i,p}(u)$ i $N_{j,q}(v)$ su neracionalne B-splajn osnovne funkcije definirane na vektorima čvora

$$U = \{0, \ldots, 0, u_{p+1}, \ldots, u_{r-p-1}, 1, \ldots, 1\} \tag{1.16}$$

$$V = \{0, \ldots, 0, u_{q+1}, \ldots, u_{q-s-1}, 1, \ldots, 1\}, \tag{1.17}$$

pri čemu vrijedi $r = n + p + 1$ i $s = m + q + 1$.



Slika 1.6: NURBS ploha

```
from geomdl import NURBS
from geomdl.visualization import VisMPL

surf = NURBS.Surface()
surf.degree_u = 3
```

```
surf.degree_v = 3
surf.ctrlpts2d = ctrlpts
surf.knotvector_u = [0.0, 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 3.0, 3.0]
surf.knotvector_v = [0.0, 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 3.0, 3.0]
surf.sample_size = 30

surf.vis = VisMPL.VisSurfTriangle()
surf.render()
```

Isječak koda 1.6: Prikaz NURBS krivulje

Za sad su promatrane samo točke na plohi, no za daljnje proučavanje algoritma teselacije potrebni su i vektori normale na plohu, pa slijedi njihova definicija.

Definicija 1.3.3. *Vektor normale na plohu u točki čije su parametarske koordinate* (u_0, v_0) *je dan sljedećom jednadžbom:*

$$N_s(u,v) = \frac{\left(\frac{\delta S(u,v)}{\delta u}\right)_{u_0,v_0} \times \left(\frac{\delta S(u,v)}{\delta v}\right)_{u_0,v_0}}{\left|\left(\frac{\delta S(u,v)}{\delta u}\right)_{u_0,v_0} \times \left(\frac{\delta S(u,v)}{\delta v}\right)_{u_0,v_0}\right|}, \tag{1.18}$$

gdje je $\frac{\delta S(u,v)}{\delta u}$ *tangencijalan vektor na plohu u smjeru u, a* $\frac{\delta S(u,v)}{\delta v}$ *tangencijalan vektor na plohu u smjeru v.*

Poglavlje 2

Teselacija

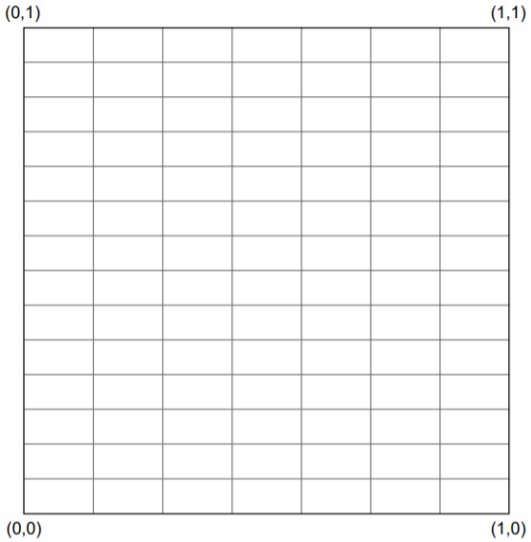
2.1 Uniformna i adaptivna teselacija

Neformalno, teselaciju možemo definirati kao podjelu ravnine u diskretne plošne elemente koji se dodiruju i prekrivaju ravninu. U ovom radu ti elementi su trokuti, budući da se radi upravo o teselaciji trokutima. Završni proces pretvaranja 3D modela u dvodimenzionalnu sliku naziva se renderiranje.
Slijedi formalna definicija teselacije plohe Ω.

Definicija 2.1.1. *Teselacija plohe Ω definira se kao skup poligonalnih područja čija je unija cijela ravnina i čiji se unutrašnji dijelovi ne sijeku. Za teselaciju se kaže da je dobro poravnata ako se bilo koja dva područja susreću ili u zajedničkom vrhu ili u zajedničkom rubu ili uopće ne. Regularna teselacija je teselacija čija se poligonalna područja podudaraju s regularnim poligonom.*

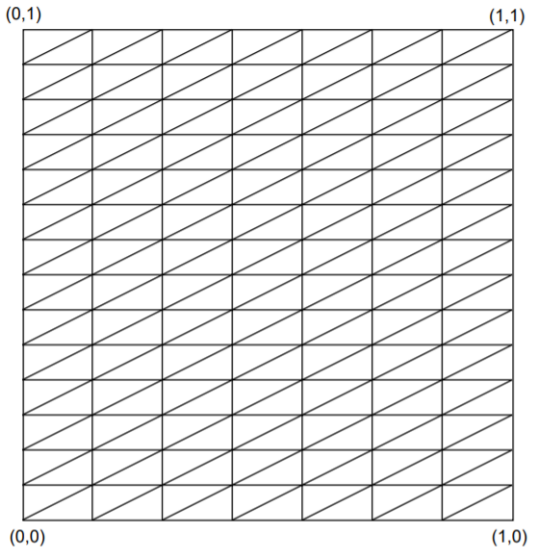
NURBS plohe koriste se za renderiranje i teselaciju raznih modela. No, gotovo svi takvi algoritmi zahtijevaju pretvaranje NURBS ploha u Bézierove plohe, budući da je složenost formulacije Bézierovih ploha manja. Slijedi da su i pripadni algoritmi jednostavniji. Prilikom pretvorbe plohe iz NURBS prikaza u Bézierov prikaz, NURBS ploha je podijeljena u skup Bézierovih ploha. Tako generirane Bézierove plohe nazivaju se "zakrpe" (eng. patch). Jedna od glavnih metoda za renderiranje ploha je teselacija. Teselacija koja se izvodi pomoću ove metode može biti uniformna ili adaptivna unutar jedne zakrpe (eng. interpatch-adaptiv). Ako se radi o uniformnoj teselaciji, onda je razlučivost mreže jednaka za cijelu figuru. Kod adaptivne teselacije, svaka zakrpa je teselirana sa svojom vlastitom razlučivosti. U ovom poglavlju bit će obrađeni neki od algoritama teselacije dostupni u literaturi [7].
Gotovo svi algoritmi teselacije Bézierovih ploha teseliraju plohe u parametarskoj ravnini i izvršavaju sljedeće korake:

1. Podjela parametarske ravnine u $n_u \times n_v$ ćelija veličine $\frac{1}{n_u} \times \frac{1}{n_v}$.
   Parametri n_u i n_v mogu biti jednaki za sve zakrpe u uniformnoj teselaciji, a u adaptivnoj teselaciji mogu ovisiti o plohi.



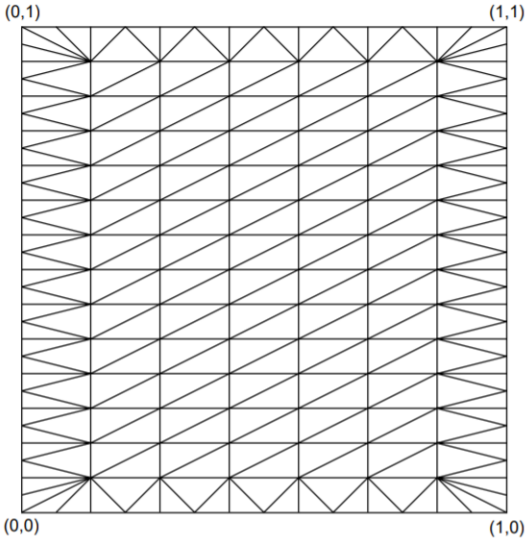
Slika 2.1: Podjela parametarske ravnine

2. Generiranje trokuta.
   Kod uniformne teselacije, svaka ćelija je podijeljena na dva trokuta.



Slika 2.2: Uniformna teselacija

U slučaju adaptivne teselacije (n_u i n_v su različiti za svaku Bézierovu zakrpu), potreban je dodatan korak kako bi se izbjegle pukotine na zajedničkim rubovima između zakrpi. U ovom koraku, četiri granične zakrpe smatraju se kao četiri pojedinačne Bézierove krivulje i razdjeljuju se neovisno jedna od druge. Na taj način, ćelije unutar jedne zakrpe se transformiraju u trokute kao kod uniformne teselacije, a rubne zakrpe generiraju trokute na način kako je prikazano na sljedećoj slici.
Individualno particioniranje za svaku graničnu krivulju osigurava da se zajednički rubovi teseliraju s istim brojem točaka, izbjegavajući pukotine i defekte ploha u završnoj teselaciji.



Slika 2.3: Adaptivna teselacija

2.2 Potpuno adaptivna teselacija NURBS ploha

Potpuno adaptivna teselacija obavlja teselaciju Bézierovih ploha s promjenjivom razlučivosti unutar pojedine plohe. To omogućuje upotrebu mreža s manjim brojem trokuta za istu kvalitetu vizualizacije. Metoda obavlja teselaciju u dva koraka:

1. dobiva se početna gruba teselacija, opisana u prošlom odjeljku;
2. početna mreža se dalje obrađuje kako bi podijelila one dijelove koji ne aproksimiraju izvornu sliku.

```
Calculate_initial_mesh;
List=initial_mesh;
while(List!=empty) {
    Extract_triangle(List);
    Test(edge1,edge2,edge3);
    if(test=TRUE)
        render_triangle;
    else {
        subdivide_triangle;
        List←new_triangles;
    }
}
```

Slika 2.4: Potpuno adaptivna teselacija - algoritam

Početni korak algoritma je kreiranje liste koja pohranjuje trokute iz inicijalne teselacije. Svaki trokut je određen s tri vrha, a svaki vrh s po osam koordinata: dvije parametarske koordinate (u, v), tri koordinate prostora (x, y, z) i tri koordinate normalnog vektora na plohu (n_x, n_y, n_z). Nakon što se pohrani početna mreža i pokrene petlja, u svakoj iteraciji petlje se obrađuje jedan trokut iz liste. Provodi se test za svaki rub trokuta. U slučaju da jedan ili više rubova ne prođe test, tj. kvaliteta teselacije nije dovoljno dobra za analizirano područje, trokut se dijeli na manje dijelove, a dobiveni trokuti se pohranjuju u listu za buduću obradu.
Ako sva tri ruba prođu test, trokut se šalje na renderiranje. Iteriranje kroz petlju se nastavlja dok lista nije prazna, a to onda znači da je cijela jedna zakrpa renderirana.
U sljedećim odlomcima detaljnije su objašnjeni pojedini dijelovi algoritma.
Za implementaciju testova korištena je programski jezik Python i već opisana biblioteka geomdl. Implementacija teselacije će biti prikazana na Bézierovoj plohi danoj u primjeru 1.1.5.

Inicijalna teselacija

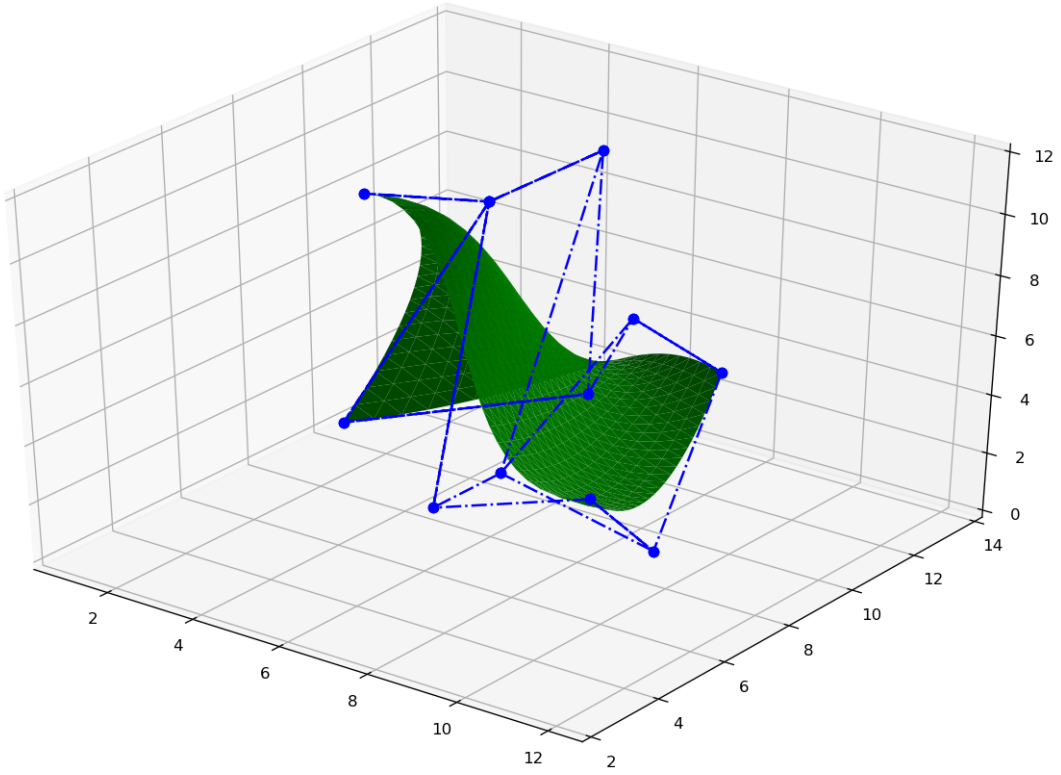
Kako bismo dobili inicijalnu mrežu, izvršena je podjela parametarske ravnine, na način kako je objašnjeno u prethodnom poglavlju. Podjela ravnine može biti uniformna, tj. jednaka za svaku zakrpu, ili adaptivna unutar svake zakrpe, tj. različita za svaku zakrpu.
Ako je inicijalna mreža uniformna, parametarske koordinate su jednake za svaku zakrpu, pa je teselacija odmah izvršiva.
Ako je inicijalna teselacija adaptivna unutar pojedinih dijelova plohe, onda broj trokuta

svake zakrpe ovisi o točki gledišta i o obliku plohe. Kao i kod slučaja uniformne teselacije, prostorne koordinate i normalni vektori izračunati su kroz dane jednadžbe.
Za implementaciju inicijalne teselacije, poziva se metoda tessellate() biblioteke geomdl. Time je stvorena mreža od ukupno 1152 trokuta.

```
1 surf.tessellator = tessellate.TriangularTessellate()
2 surf.tessellate()
```

Isječak koda 2.1: Inicijalna teselacija

Slika 2.5 prikazuje graf Bézierove plohe nakon inicijalne teselacije.



Slika 2.5

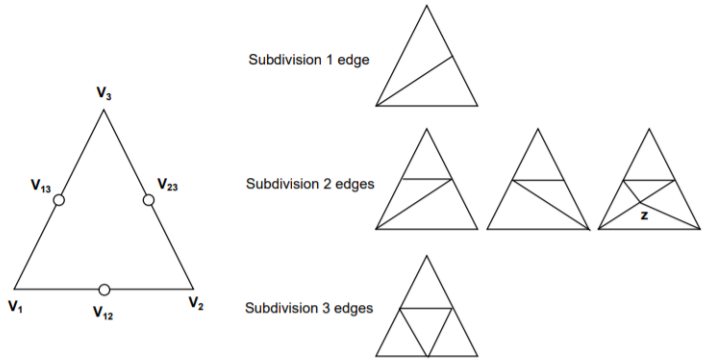
Potpuno adaptivna podjela na manje dijelove

Uzimajući inicijalnu mrežu kao početnu točku, bridovi trokuta se ispituju pomoću testova. Prilikom svakog testa se donosi odluka o ubacivanju novog vrha u brid. Ako su jedan ili više bridova podijeljeni, to rezultira dvama ili više trokuta koji se ponovno analiziraju.

Postupak se ponavlja sve dok ne prođu svi testovi.
Dakle, testiranje svakog brida obavlja se u dva koraka:

1. izračunavanje novog vrha;
2. provođenja testa koji odlučuje hoće li se umetnuti novi vrhovi.

Parametarske koordinate novog vrha izračunate su pomoću krajeva bridova. Nova točka je središnja točka u parametarskoj ravnini ($V_{1,2} = \frac{(u_1,v_1)+(u_2,v_2)}{2}$). Ostale koordinate izračunavaju se pomoću Bézierovih jednadžbi.
Svaki trokut treba biti testiran kako bi se odlučilo o ubacivanju novih vrhova. Kako bi se izračunala udaljenost vrha od plohe, testovi koriste informacije o bridu i susjednim bridovima. Rezultat testa je tipa *boolean*, što znači da vraća vrijednost *true* ili $false$. Testovi daju rezultat za svaki brid trokuta. Ovisno o tri dobivene vrijednosti, trokut je podijeljen sljedećom shemom:



Slika 2.6: Podjela trokuta

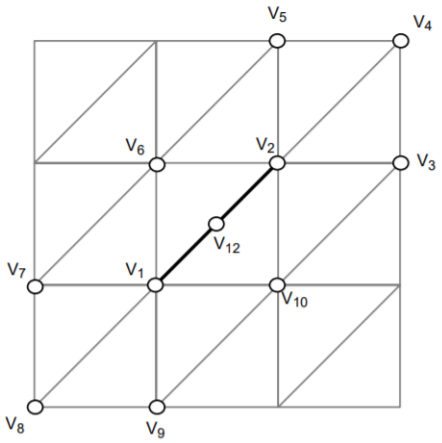
U slučaju podjele dvaju bridova dane su dvije mogućnosti, ovisno o tome umeće li se dodatni vrh ili ne.
Sljedeće, predstavljeni su testovi koji se koriste u adaptivnoj podjeli mreža. Podijeljeni su u dvije grupe:

1. testovi koji koriste normalne vektore rubnih vrhova i susjeda;
2. testovi koji koriste informacije o glatkoći (eng. flatness) mreže na određenom području.

Oba testa mjere je li mreža dovoljna glatka u području oko analiziranog ruba ili ne.

Normalni testovi uspoređivanja

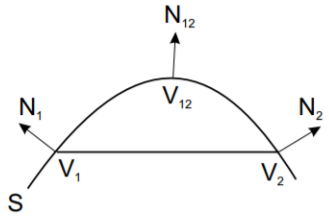
Normalni testovi uspoređivanja izračunavaju vektor odstupanja između analiziranih točaka i njihovih susjeda. Svrha korištenja ovih testova je osigurati da normalni vektori susjednih vrhova nisu previše različiti. Tako se izbjegava loše sjenčanje u renderiranoj mreži.
Postoji nekoliko mogućnosti uspoređivanja. Predloženi testovi se razlikuju od onih koji koriste samo informacije o lokalnim bridovima (vrhovi V_1, V_2, V_{12} na slici) do onih koji, generalizirajući takve osnovne testove, koriste informacije o svim susjedima prvog reda (vrhovi V_1 do V_{10} na slici).



Slika 2.7: Susjedi prvog reda brida $V_1 - V_2$

U nastavku je opisano nekoliko testova uspoređivanja.

Testiranje jednog brida (eng. one-edge normal test)



Slika 2.8: Testiranje jednog brida

Testiranje jednog brida korištenjem lokalnih informacija o bridu je najjednostavniji test. Osim koordinata vrhova (V_1, V_2 i V_{12}), koriste se i informacije o normalnim vektorima (N_1, N_2 i N_{12}). Odabire se granična vrijednost t_1, a test se onda temelji na sljedećem:

$$Test = (|N_1 - N_{12}| > t_1) \vee (|N_2 - N_{12}| > t_1). \tag{2.1}$$

Ako je zadovoljena jedna od nejednakosti iz prethodno napisanog testa, znači da test nije prošao i brid $V_1 - V_2$ je podijeljen umetanjem novog vrha V_{12}.
Slijedi prikaz implementacije ovog testa korištenjem biblioteke geomdl.

```
def one_edge_normal_test(vertex1, vertex2, value):
    middle_point = (vertex1+vertex2)/2
    vertex1_norm = surf.normal(vertex1.uv)[1]
    vertex2_norm = surf.normal(vertex2.uv)[1]
    middle_point_norm = surf.normal(middle_point.uv)[1]

    result1 = ((vertex1_norm[0] - middle_point_norm[0]), (vertex1_norm
    [1] - middle_point_norm[1]), (vertex1_norm[2] - middle_point_norm
    [2]))
    result1_abs = math.sqrt(pow(result1[0], 2) + pow(result1[1], 2) +
    pow(result1[2], 2))

    result2 = ((vertex2_norm[0] - middle_point_norm[0]), (vertex2_norm
    [1] - middle_point_norm[1]), (vertex2_norm[2] - middle_point_norm
    [2]))
    result2_abs = math.sqrt(pow(result2[0], 2) + pow(result2[1], 2) +
    pow(result2[2], 2))

    if result1_abs > value or result2_abs > value:
        return False

    return True
```

Isječak koda 2.2: Testiranje jednog brida

Na ovaj način, dobiveno je ukupno 2062 trokuta.

Komplementarni test jednog brida (eng. one-edge complementary test)

Ovaj test je proširenje običnog testa jednog brida objašnjenog u prethodnom pododjeljku. Uz provjere u testu (2.1), komplementarni test radi i dodatnu usporedbu između normalnih vektora rubnih vrhova:

$$Test = (|N_1 - N_2| > t_2), \tag{2.2}$$

pri čemu je $t_2 \neq t_1$.
Kombiniranjem ovog uvjeta s testom (2.1), dobiva se da je komplementarni test oblika:

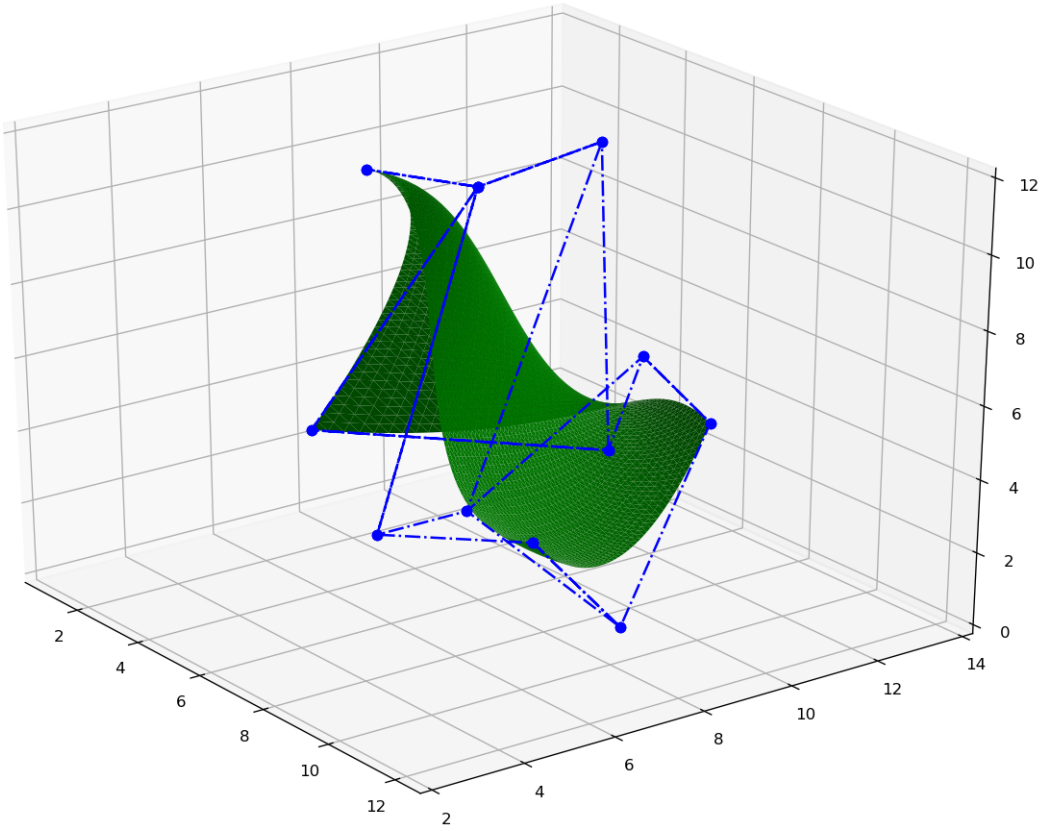
$$Test = (|N_1 - N_{12}| > t_1) \vee (|N_2 - N_{12}| > t_1) \vee (|N_1 - N_2| > t_2), \tag{2.3}$$

tj. ako je jedna od nejednakosti veća od svoje granične vrijednosti, onda se ubacuje novi vrh u brid.
Slijedi prikaz implementacije komplementarnog testa jednog brida.

```
def one_edge_complementary_test(vertex1, vertex2, value1, value2):
    test = one_edge_normal_test(vertex1, vertex2, value1)
    if not test:
        return False

    vertex1_norm = surf.normal(vertex1.uv)[1]
    vertex2_norm = surf.normal(vertex2.uv)[1]

    result = ((vertex1_norm[0] - vertex2_norm[0]), (vertex1_norm[1] -
    vertex2_norm[1]), (vertex1_norm[2] - vertex2_norm[2]))
    result_abs = math.sqrt(pow(result[0], 2) + pow(result[1], 2) + pow(
    result[2], 2))

    if result_abs > value2:
        return False

    return True
```

Isječak koda 2.3: Komplementarni test jednog brida

Budući da je ovaj test proširenje prethodnog testa, očekivano je da će broj generiranih trokuta biti veći nego u prošlom testu. Rezultat od 2290 trokuta potvrđuje tu pretpostavku. Slika 2.9 prikazuje graf Bézierove krivulje s generiranih 2290 trokuta.



Slika 2.9

Test dvaju trokuta (eng. two triangle test)

Osim informacija o bridu kojeg ispitujemo, ovaj test koristi i informacije o dodatna dva vrha, koji su vrhovi dvaju trokuta koji dijele zajednički brid (na slici 2.7 vrhovi V_6 i V_{10}). Test se onda može zapisati u sljedećem obliku:

$$Test = |N_{12} - N'| > t_3, \tag{2.4}$$

pri čemu je t_3 granična vrijednost, a $N' = \frac{N_1+N_2+N_6+N_{10}}{4}$.

Test susjeda prvog reda (eng. first order neighbour test)

Ovaj test koristi informacije o zadanom bridu i o susjedima prvog reda. Test susjeda prvog reda zadan je sljedećim izrazom:

$$Test = |N_{12} - N''| > t_3, \tag{2.5}$$

gdje je t_3 granična vrijednost, n broj susjeda prvog reda, a $N'' = \sum_{i=1}^{n} \frac{N_i}{n}$.

Testovi glatkoće

Testovi glatkoće (engl. flat tests) koriste prostorne koordinate vrhova trokuta za testiranje glatkoće mreže. Ako je područje blizu brida glatko, umetanje novog vrha ne povećava kvalitetu mreže.
U sljedećim odjeljcima predstavljeno je nekoliko metoda za ispitivanje glatkoće mreže u području blizu vrha. Najjednostavnije metode koriste samo informaciju o promatranom vrhu, dok složenije metode koriste informacije o susjednim vrhovima.

Test udaljenosti vektora (eng. vector deviation flat test)

Kako bi osigurao da je mreža dovoljno glatka, test udaljenosti vektora mjeri udaljenost između novog vrha i brida trokuta na koji bi se novi vrh mogao umetnuti.
Ovaj test računa normalizirani vektor $|V_1 - V_2|$ i skalarni produkt tog vektora s vektorima $|V_{12} - V_1|$ i $|V_{12} - V_2|$. Na taj način, premda se udaljenost od vrha do brida ne mjeri izravno, izračunavamo odstupanje između vektora koje upućuje na novi vrh.
Test možemo prikazati sljedećim izrazom:

$$Test = \|V_{12} - V_1| \cdot |V_1 - V_2\| < t_1 \vee \|V_{12} - V_2| \cdot |V_1 - V_2\| < t_1. \tag{2.6}$$

Ako je zadovoljena jedna od nejednakosti iz prethodnog izraza, brid se dijeli ubacivanjem novog vrha.
U nastavku je dana implementacija opisanog testa. Kao rezultat teselacije dobivena su 1455 trokuta.

```
1  def vector_deviaton_flat_test(vertex1, vertex2, value):
2      middle_point = (vertex1+vertex2)/2
3
4      result1 = middle_point - vertex1
5      result2 = vertex1 - vertex2
6      result3 = middle_point - vertex2
7
8      result_abs1 = abs(result1[0] * result2[0] + result1[1] * result2[1]
       + result1[2] * result2[2])
9      result_abs2 = abs(result3[0] * result2[0] + result3[1] * result2[1]
       + result3[2] * result2[2])
10
11     if result_abs1 < value or result_abs2 < value:
12         return False
13
14     return True
```

Isječak koda 2.4: Test udaljenosti vektora

Lokalni test glatkoće (eng. local flat test)

Drugi način za testirati glatkoću oko vrha je usporediti tangencijalan vektor s vektorom normale na analizirani vrh. Ako je područje oko vrha glatko, vektori su međusobno okomiti, tj. skalarni produkt je nula. Upravo to radi lokalni test glatkoće.
Test prvo računa i normalizira vektor $U = |V_1 - V_2|$, te računa skalarni produkt vektora U s vektorima normale na vrhove brida, označimo ih N_1 i N_2. Zatim se oba skalarna produkta uspoređuju s graničnom vrijednosti. Ako je jedan od njih veći od granične vrijednosti, promatrani brid se dijeli ubacivanjem novog vrha.
Slijedi prikaz implementacije lokalnog testa glatkoće. Broj generiranih trokuta u ovom slučaju je 2302.

```
def local_flat_test(vertex1, vertex2, value):
    vector = vertex1 - vertex2
    vector_abs = math.sqrt(pow(vector[0], 2) + pow(vector[1], 2) + pow(
    vector[2], 2))


    vertex1_norm = surf.normal(vertex1.uv)[1]
    vertex2_norm = surf.normal(vertex2.uv)[1]

    vertex1_norm_abs = math.sqrt(pow(vertex1_norm[0], 2) + pow(
    vertex1_norm[1], 2) + pow(vertex1_norm[2], 2))
    vertex2_norm_abs = math.sqrt(pow(vertex2_norm[0], 2) + pow(
    vertex2_norm[1], 2) + pow(vertex2_norm[2], 2))

    if vertex1_norm_abs*vector_abs > value or vertex2_norm_abs*
    vector_abs > value:
        return False

    return True
```

Isječak koda 2.5: Lokalni test glatkoće

Test glatkoće susjeda prvog reda (eng. first order neighbours flat test)

Prethodni test je poseban slučaj općenitijeg testa koji koristi informacije o susjedima prvog reda za analizu bridova.
Ovaj test koristi tzv umbrella operator, čija definicija slijedi.

Definicija 2.2.1. *Umbrella operator definiran je sljedećim izrazom:*

$$U(v) = \frac{1}{n} \sum_{i=0}^{n-1} V_i - V, \tag{2.7}$$

gdje je V vrh čija se glatkoća analizira, a V_i su susjedni vrhovi vrha V.

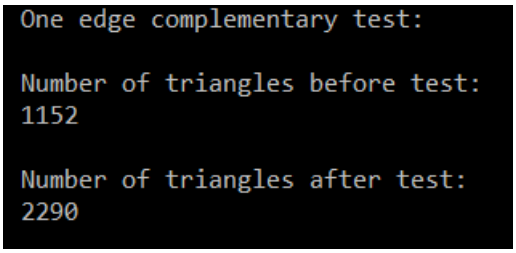
Test glatkoće najprije računa i normalizira umbrella operator nad točkama $V = V_1$ i $V = V_2$. Neka su pripadni umbrella operatori označeni s U_1 i U_2. Zatim se računa skalarni produkt normaliziranih umbrella operatora U_1 i U_2 s pripadnim vektorima normale N_1 i N_2. Tako dobiveni skalarni produkti se uspoređuju s graničnom vrijednošću. Ako je vrijednost jednog skalarnog produkta veća od granične vrijednosti, ubacuje se novi vrh.

Slijedi implementacija opisanog algoritma teselacije. U ovom primjeru korišten je komplementarni test jednog brida.

```
print("One edge complementary test")
print("Number of triangles before test:")
print(counter)

triangles_list = surf.tessellator.triangles
while triangles_list:
    triangle = triangles_list.pop(0)

    try:
        test = one_edge_complementary_test(triangle.vertices[0],
    triangle.vertices[1], triangle.vertices[2], 0.02, 0.01)
        if test:
            pass
        else:
            new_triangle1 = elements.Triangle()
            new_triangle1.add_vertex(triangle.vertices[0])
            new_triangle1.add_vertex(triangle.vertices[2])
            new_triangle1.add_vertex((triangle.vertices[0] + triangle.
    vertices[1])/2)

            new_triangle2 = elements.Triangle()
            new_triangle2.add_vertex(triangle.vertices[1])
            new_triangle2.add_vertex(triangle.vertices[2])
            new_triangle2.add_vertex((triangle.vertices[0] + triangle.
    vertices[1])/2)

            triangles_list.append(new_triangle1)
            triangles_list.append(new_triangle2)

            counter += 1

    except:
        print("ValueError: The magnitude of the vector is zero")

```

```
32
33 print("Number of triangles after test:")
34 print(counter)
```

U nastavku je priložen ispis iz programa za komplementarni test jednog brida.



Slika 2.10: Izvještaj o teselaciji

Poglavlje 3

Teselacija u sustavu OpenGL

3.1 Uvod u OpenGL

OpenGL [5] predstavlja sučelje za programiranje grafičkih aplikacija (eng. application programming interface (API)). Osnovan je 1992. godine, te postaje na široko industrijski korišten. Razvojni programeri mogu koristiti OpenGL na svim platformama. Odlikuje se odličnom podrškom i opširnom dokumentacijom, što ga čini pogodnim za učenje i korištenje. OpenGL potiče inovaciju i ubrzava razvoj aplikacija korištenjem širokog skupa renderiranja, mapiranja tekstura, specijalnih efekata i drugih snažnih funkcija vizualizacije.

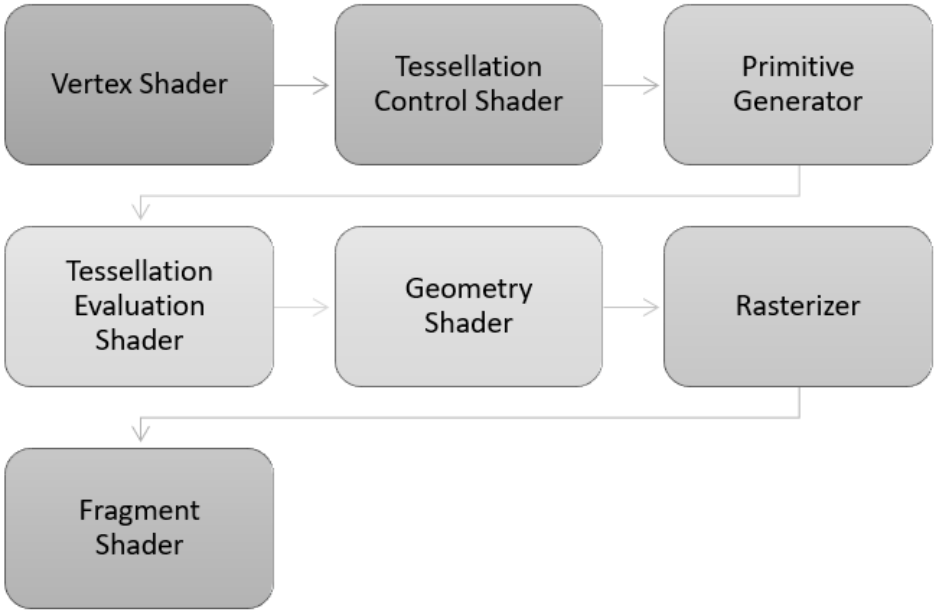
Slika 3.1: OpenGL logo

Nakon godina industrijskih i akademskih istraživanja, teselacija se pojavljuje kao nova značajka u verziji OpenGL 4.x. Pozadina ove značajke je matematička, te se tiče već objašnjenih Bézierovih krivulja i ploha. Temeljni problem s kojim se teselacija nosi je statička priroda 3D modela u pogledu njihovih detalja. Kada se pogleda kompleksni model kao što je na primjer ljudsko lice, potrebno je upotrijebiti vrlo detaljan model koji će obratiti pažnju na sve potrebne detalje (npr. bore, akne, pjegice). Takav model će generirati više trokuta i trošiti više računalne snage za obradu.
Jedan od mogućih načina rješavanja ovog problema pomoću postojećih značajki OpenGL-a je generiranje istog modela na više razina detalja (eng. Level of Details). Razine detalja mogu biti sljedeće: vrlo detaljna, prosječna i niska razina.
Teselacija u OpenGL-u predstavlja fazu obrade vrhova, pri čemu su plohe zadane vrhovima podijeljene na manje primitivne oblike, u slučaju ovog rada, trokute. Za taj postupak

potrebna su dva sjenčara i fiksna funkcija stanja.
U nastavku, prvo je prikazano koje faze su potrebne za izvođenje teselacije u sustavu OpenGL, a zatim je pobliže objašnjena svaka od faza [4, 3].

3.2 Grafički protočni sustav

U ovom poglavlju je objašnjen grafički protočni sustav u OpenGL-u.
Prva faza koju zahtijeva OpenGL u postupku teselacije je sjenčar vrhova (eng. Vertex Shader). To je faza u postupku teselacije koja obrađuje vrhove ovisno o željama razvojnog programera. Ulazni i izlazni podatak sjenčara vrhova je uvijek točno jedan vrh.
Nakon sjenčara vrhova slijede faze odgovorne za teselaciju. Prva faza sjenčanja odgovorna za teselaciju je sjenčar za kontrolu teselacije (eng. Tessellation Control Shader). Fiksna funkcija stanja koja slijedi nakon sjenčara za kontrolu teselacije naziva se primitivni generator (eng. Primitive Generator). Druga faza sjenčanja odgovorna za teselaciju naziva se sjenčar za evaluaciju teselacije (eng. Tessellation Evaluation Shader). Tri navedene faze detaljnije su opisane u nastavku.
Nakon sjenčara za evaluaciju teselacije slijedi geometrijski sjenčar (eng. Geometry Shader), zatim rasterizacija (eng. Rasterization) i fragmentni sjenčar (eng. Fragment Shader).
Geometrijski sjenčar je opcionalan. To je dio programa definiran od razvojnog programera zadužen za procesiranje svake dolazne primitivne plohe (trokuta). Nakon procesiranja, vraća nula ili više trokuta. Moguće je maknuti neke trokute ili ih teselirati i vratiti više izlaznih podataka za jedan ulazni. Geometrijski sjenčar može pretvoriti primitivnu plohu iz jednog oblika u drugi, npr. iz trokuta u kvadrat, ali to nije slučaj u ovom radu.
Sljedeća faza je rasterizacija. Primitivni oblici su rasterizirani, odnosno dijeljeni na manje diskretne elemente, redom kojim su dani. Rezultat rasterizacije je niz fragmenata. Fragment je skup stanja korišten za izračunavanje konačnih podataka za prikaz pixela na ekranu.
Svaki podatak procesiran postupkom rasterizacije je zatim procesuiran od strane fragmentnog sjenčara. Izlazni podatak su detalji o boji i dubini svake vrijednosti.
Fragmentni sjenčar je opcionalan u postupku teselacije.
Dijagram (3.2) prikazuje opisani grafički protočni sustav u OpenGL-u.



Slika 3.2: Struktura teselacije u OpenGL-u

Sjenčar za kontrolu teselacije

Sjenčar za kontrolu teselacije radi s kontrolnim točkama, opisanima u prvom poglavlju rada. Kontrolne točke ne definiraju pravilni mnogokut kao što su trokut ili kvadrat, već definiraju geometrijsku plohu. Sjenčar za kontrolu teselacije kao ulazni podatak prima plohu određenu kontrolnim točkama. Izlazni podatak je također geometrijska ploha, no sjenčar za kontrolu teselacije razvojnom programeru omogućuje transformaciju kontrolnih točaka, kao i njihovo dodavanje ili uklanjanje.
Sljedeća bitna zadaća sjenčara za kontrolu teselacije je računanje razine teselacije (eng. Tessellation Level). Razina teselacije određuje koliko je trokuta potrebno generirati po jednoj plohi. Algoritam koji određuje razinu teselacije bira programer. Različite plohe, ovisno o svojim karakteristikama, mogu imati međusobno različite razine teselacije.
Sjenčar za kontrolu teselacije je opcionalan u postupku teselacije. Ako nije aktivan u trenutnom izvođenju programa, podaci o plohi se prenose izravno od sjenčara vrhova. U tom slučaju su podaci o razini teselacije jednaki unaprijed postavljenim vrijednostima.

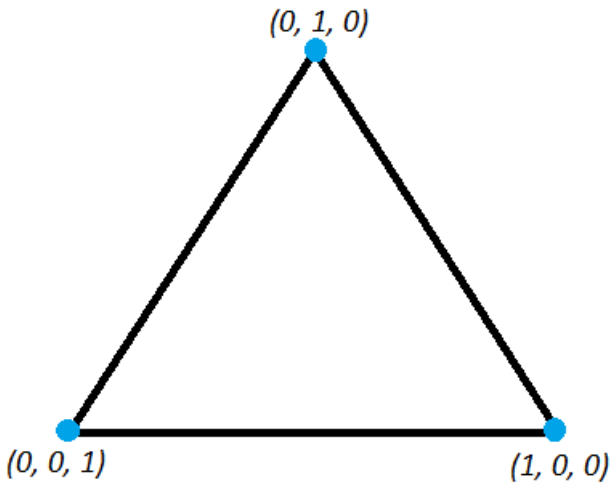
Primitivni generator

Nakon što sjenčar za kontrolu teselacije završi s radom, inicijativu preuzima primitivni generator, čiji je zadatak podjela plohe na trokute. U nastavku su navedeni čimbenici o

kojima ovisi primitivni generator.

- Razina teselacije, koju kao izlazni podatak daje kontrola sjenčanja teselacije.
- Razmak između teseliranih vrhova, koji može poprimiti sljedeće vrijednosti: `equal_spacing`, `fractional_even_spacing`, ili `fractional_odd_spacing`.
- Oblik primitivne plohe na kojoj se temelji teselacija. To može biti trokut, kvadrat ili linija. U ovom radu će se proučavati samo trokuti.

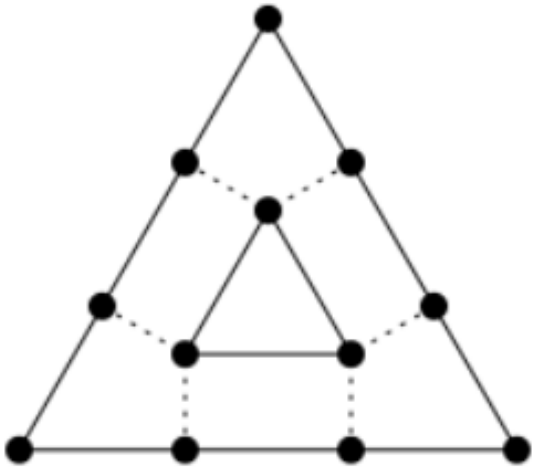
Primitivni generator nema pristup plohi koju kao izlazni podatak daje sjenčar za kontrolu teselacije. Njegov zadatak je podijeliti primitivnu plohu, u slučaju ovog rada trokut, na manje dijelove. Trokut je zadan baricentričnim koordinatama. Baricentrične koordinate omogućuju definiranje pozicije točke unutar trokuta kao linearnu kombinaciju težina triju vrhova trokuta. Kako se točka unutar trokuta više približava određenom vrhu, tako se povećava težina tog vrha, dok se težine ostalih vrhova smanjuju. Ako se točka nalazi baš na vrhu, tada je težina tog vrha 1, a težine preostalih vrhova su 0. Zato su baricentrične koordinate vrhova dane sljedećim vrijednostima: $(0, 0, 1)$, $(0, 1, 0)$ i $(0, 0, 1)$. Zanimljivo svojstvo baricentričnih koordinata je to da je zbroj individualnih komponenti jedne baricentrične koordinate uvijek jednak 1.
Na slici (3.3) je prikazan trokut koji je zadan baricentričnim koordinatama.



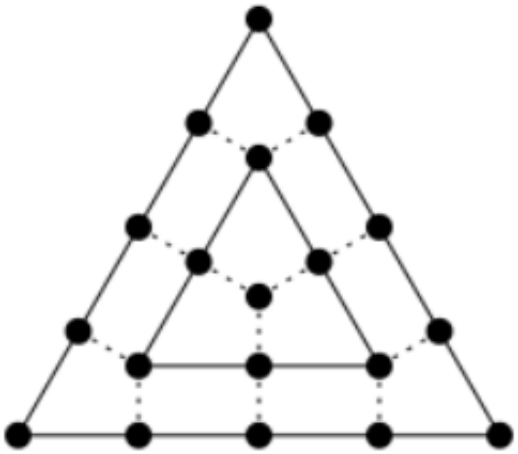
Slika 3.3: Trokut zadan baricentričnim koordinatama

Dakle, ovisno o vrijednosti razine teselacije, primitivni generator generira skup točaka unutar trokuta. Svaka ta točka je definirana svojim baricentričnim koordinatama, koje

određuju poziciju vrha unutar početnog trokuta. Primitivni generator kao izlazni podatak daje točku ili trokut. U slučaju da je izlazni podatak trokut, primitivni generator povezuje sve točke tako da je cijelo naličje trokuta teselirano manjim trokutima.
Više puta je spomenuto da generiranje novih točaka ovisi o razini teselacije. Razina teselacije dijeli se na vanjsku (eng. outer tessellation level) i unutarnju (eng. inner tessellation level). Vanjska razina teselacije definira teselaciju za vanjske bridove početne apstraktne plohe. Unutarnja razina teselacije definira teselaciju unutar apstraktne plohe. Općenito, vanjsku razinu teselacije određuje četverodimenzionalni vektor, dok je unutarnja razina teselacije određena dvodimenzionalnim vektorom. U slučaju teselacije trokutima, koriste se tri vrijednosti za vanjsku razinu teselacije i jedna vrijednost za unutarnju razinu teselacije. Svaki brid trokuta odgovara jednoj vanjskoj razini teselacije. Zadatak vanjske teselacije je podijeliti početni brid na više manjih bridova, čiji je zbroj duljina jednak duljini početnog brida. Međusoban odnos pojedinih duljina manjih bridova ovisi o „spacing“ parametru kojeg definira sjenčar za kontrolu teselacije. U ovom radu koristi se `equal_spacing`. Zbog ovog svojstva, svi teselirani bridovi imaju jednaku duljinu.
Način na koji funkcionira unutarnja teselacije je možda manje intuitivan, jer broj koncentričnih trokuta koje dobivamo unutar početnog trokuta ne odgovara unutarnjoj razini teselacije.
Unutarnja teselacija se primjenjuje na vanjski trokut prije vanjske teselacije. Vanjski bridovi podijeljeni su na onoliko manjih bridova jednake duljine koliko iznosi razina unutarnje teselacije. Za svaki kut trokuta se uzimaju dva susjedna umetnuta vrha na bridu, te se promatra sjecište okomica koje su spuštene na bridove iz tih vrhova. Okomite linije spuštene s preostalih umetnutih vrhova definiraju vrhove koji će dijeliti novonastali trokut na manje dijelove. Taj postupak se ponavlja na unutarnjim trokutima, dok ne nastupi završni uvjet. Podjela staje kad trokut kojeg dijelimo nema više vrhova na bridovima za spuštanje okomica i podjelu, ili kad taj trokut na svakom bridu ima točno jedan vrh i spuštanjem okomica se dobiva točno jedan vrh.
Slijedi da je broj koncentričnih trokuta dobiven ovim algoritmom jednak najvećem cijelom od polovice razine unutarnje teselacije. Slika (3.4) prikazuje podjelu trokuta u slučaju kad je unutarnja razina teselacije jednaka 2, a slika (3.5) u slučaju kad je unutarnja razina teselacije jednaka 4.

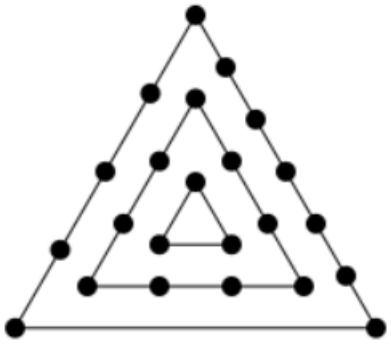


Slika 3.4: Inner Tess = 3



Slika 3.5: Inner Tess = 4

Nakon što završi algoritam podjele trokuta unutarnjom teselacijom, primjenjuje se gore opisan postupak vanjske teselacije.
Slika (3.6) prikazuje podjelu trokuta u slučaju kad je razina unutarnje teselacije jednaka 5, a razine vanjske teselacije su različite za svaki brid i iznose 4, 1 i 6. Prvo je izvršena podjela na koncentrične trokute unutarnjom teselacijom, a zatim je primijenjena vanjska teselacija na vanjske bridove polaznog trokuta.



Slika 3.6

Sjenčar za evaluaciju teselacije

Sljedeća faza u postupku teselacije u OpenGL-u je sjenčar za evaluaciju teselacije.
Kako primitivni generator nema pristup plohi koju kao rezultat daje sjenčar za kontrolu teselacije, potrebno je da netko drugi pristupi tom rezultatu. Zadaća sjenčara za evaluaciju teselacije je izračunati stvarne vrijednosti vrhova, uzimajući u obzir koordinate vrhova koje daje primitivni generator, te izlaznu plohu koju daje sjenčar za kontrolu teselacije. Primitivni generator poziva sjenčar za evaluaciju teselacije za svaku baricentričnu koordinatu. Kako sjenčar za evaluaciju teselacije ima informacije o poziciji i normali generirane plohe, koristi te informacije za generiranje vrhova. Nakon što primitivni generator izvrši evaluaciju sjenčanja za sve tri baricentrične koordinate malog trokuta, sjenčar za evaluaciju teselacije generira tri vrha i šalje ih u sljedeći stadij kao kompletan trokut.
Dakle, osnovna zadaća sjenčara za evaluaciju teselacije je pozicionirati baricentrične koordinate u polinomne koje predstavljaju plohu i izračunati poziciju novog vrha. Ako je razina teselacije viša, dobiva se više teseliranih trokuta i nakon što ih sjenčar za evaluaciju teselacije obradi, dobiva se više vrhova koji bolje predstavljaju geometrijsku plohu.

Poglavlje 4

Implementacija i rezultati

4.1 Opis rješenja

Opis korištenih alata

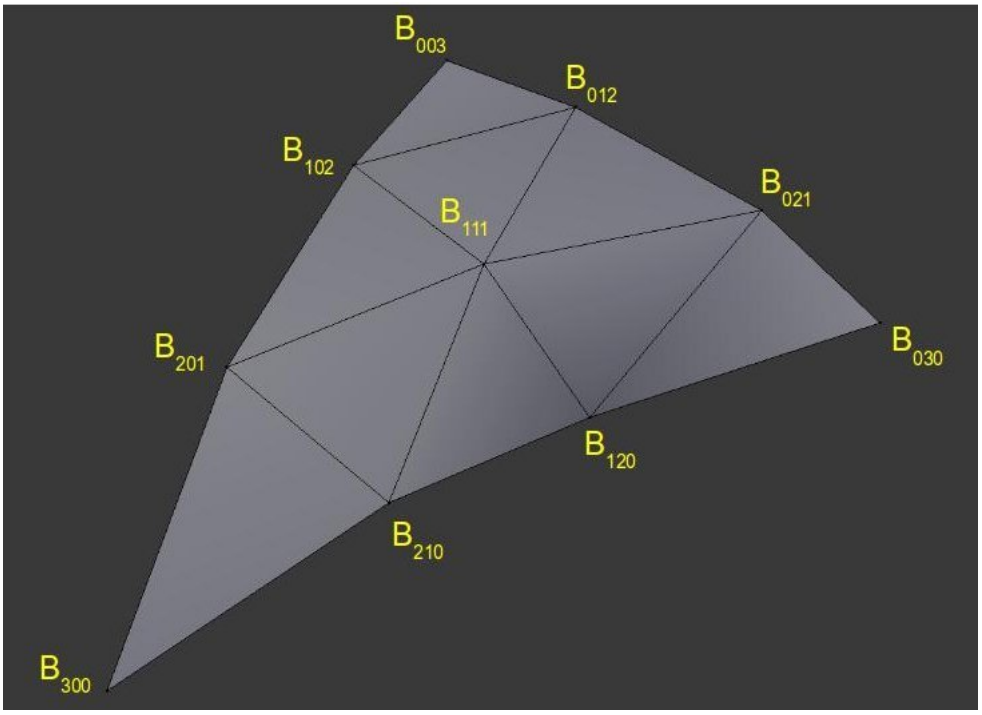
Rješenje je implementirano u programskom jeziku C++. Osnovne korištene biblioteke za rad s OpenGL-om su GLUT (OpenGL Utility Library) i GLEW (OpenGL Extension Wrangler Library). GLUT omogućuje jednostavno baratanje prozorima, event handlingom, inputom i outputom i ostalim servisima. GLEW omogućuje jednostavno korištenje ekstenzija. Jednom kad je inicijaliziran, dinamički učitava ekstenzije i omogućen im je jednostavan pristup preko jedne header datoteke.

Bézierov trokut

U prvom poglavlju dana je definicija Bézierove plohe koja je zadana jednadžbom (1.1.4). Za rješenje problema teselacije koristi se algoritam [1, 2] koji koristi poseban oblik Bézierove plohe, tzv. Bézierov trokut. Bézierov trokut je dan sljedećom jednadžbom:

$$\begin{aligned} B(u,v,w) &= \sum_{i+j+k=3} P_{ijk} \frac{3!}{i!\,j!\,k!} u^i v^j w^k \\ &= P_{300}u^3 + P_{030}v^3 + P_{003}w^3 \\ &\quad + 3P_{210}u^2w + 3P_{201}u^2w + 3P_{120}v^2u \\ &\quad + 3P_{021}v^2w + 3P_{102}w^2u + 3P_{012}w^2v \\ &\quad + 6P_{111}uvw \end{aligned} \tag{4.1}$$

Slika (4.1) prikazuje opisani Bézierov trokut.



Slika 4.1

Rješenje

U nastavku je objašnjeno rješenje problema teselacije Bézierovog trokuta, s posebnim naglaskom na fazama koje su odgovorne za teselaciju.
Postupak započinje sjenčar vrhova tako da postavlja pozicija za svaki vrh.

Sjenčar za kontrolu teselacije

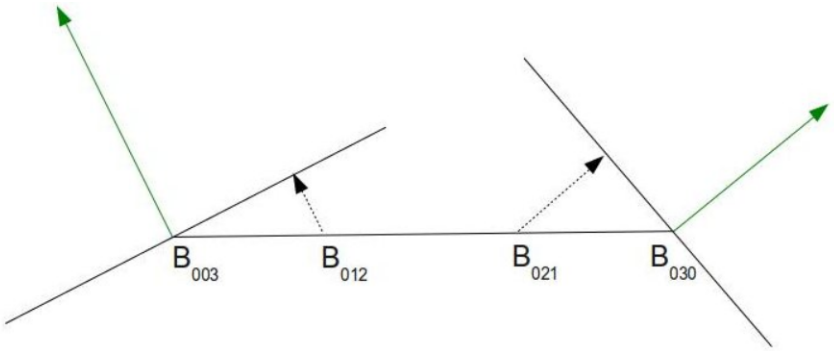
Nakon sjenčara vrhova slijedi sjenčar za kontrolu teselacije, čiji je ulazni podatak trokut zadan s tri vrha, a izlazni podatak Bézierov trokut zadan s 10 kontrolnih točaka.
Stvorena je struktura `OutputPatch` koja definira opisani Bézierov trokut zadan s 10 kontrolnih točaka. Deklarirana je varijabla tog tipa. Sjenčar za kontrolu teselacije će biti izvršen jednom za svaki Bézierov trokut i struktura će biti popunjena podacima za svih 10 točaka.
Varijabla tipa `OutputPatch` ispred deklaracije sadrži još i ključnu riječ `patch`. To znači da varijabla sadrži podatke koji se odnose na čitavu plohu, a ne samo na trenutnu izlaznu

kontrolnu točku. Kompajler koristi tu informaciju kako bi se kod koji svojim izvođenjem ažurira takvu varijablu izvodio jednom po plohi umjesto jednom po kontrolnoj točki.

```
struct OutputPatch
{
    vec3 WorldPos_B030;
    vec3 WorldPos_B021;
    vec3 WorldPos_B012;
    vec3 WorldPos_B003;
    vec3 WorldPos_B102;
    vec3 WorldPos_B201;
    vec3 WorldPos_B300;
    vec3 WorldPos_B210;
    vec3 WorldPos_B120;
    vec3 WorldPos_B111;
    vec3 Normal[3];
    vec2 TexCoord[3];
};

in patch OutputPatch outputPatch;
```

Isječak koda 4.1: Struktura Bézierove plohe

U `main` funkciji se prvo podaci o normali i teksturi preuzimaju od input plohe. Sjenčar za kontrolu teselacije zahtijeva da normale imaju jediničnu duljinu, pa zato moraju biti normalizirane. U protivnom kontrolne točke koje određuju plohu neće biti ispravno generirane.
Zatim se generira 10 kontrolnih točaka koje definiraju Bézierov trokut na način opisan u nastavku. Vrhovi Bézierovog trokuta kojeg konstruiramo su jednaki vrhovima trokuta koji je ulazni podatak za kontrolu sjenčanja. Nakon toga, svakom bridu trokuta ubačena su dva vrha koja dijele brid na tri brida jednakih duljina. Zatim je iz svakog od projiciranih vrhova spuštena okomica na ravninu definiranu najbližim vrhom početnog trokuta i njegovom normalom, kao što je prikazano na slici (4.2)



Slika 4.2

Kako bi se izračunale koordinate vrha unutar trokuta, potrebno je izračunati vektor udaljenosti između centra originalnog trokuta i aritmetičke sredine točaka ubačenih na bridove.
U prethodnom poglavlju navedeno je da je sljedeći osnovni zadatak sjenčara za kontrolu teselacije određivanje razine teselacije. U implementaciji korisnik ručno zadaje razinu teselacije, te se uzima da je vanjska razina teselacije jednaka unutarnjoj razini teselacije.

Primitivni generator i sjenčar za evaluaciju teselacije

Na početku je sjenčaru za evaluaciju teselacije potrebno zadati neke osnovne postavke nakon ključne riječi `layout`:

- `triangles` – domena s kojom radi primitivni generator
- `equal_spacing` – bridovi trokuta će biti podijeljeni na segmente jednakih duljina
- `ccw` – primitivni generator generira trokute u smjeru suprotnom od smjera kazaljke na satu (eng. counter-clockwise order)

```
1 layout(triangles, equal_spacing, ccw) in;
```

Isječak koda 4.2: Layout opcije

Sjenčar vrhova je procesirao vrhove i izračunate su pozicije i normale točaka. Sjenčar za kontrolu teselacije kao ulazni podatak uzima trokut kao Bézierovu plohu definiranu s 3 kontrolne točke, transformira je u Bézierov trokut definiran s 10 kontrolnih točaka i prosljeđuje je sjenčaru za evaluaciju teselacije. Dakle, ulazni podatak sjenčara za evaluaciju teselacije je:

```
in patch OutputPatch outputPatch;
```

Isječak koda 4.3: Ulazni podatak za evaluator sjenčanja

Primitivan generator dijeli jednakostraničan trokut na manje trokute već opisanim postupkom te poziva sjenčar za evaluaciju teselacije za svaki generirani vrh. U svakom pozivu sjenčara za evaluaciju teselacije može se pristupiti baricentričnim koordinatama preko trodimenzionalnog vektora `gl_TessCoord`.

```
float u = gl_TessCoord.x;
float v = gl_TessCoord.y;
float w = gl_TessCoord.z;
```

Isječak koda 4.4: Baricentrične koordinate

Svojstvo da baricentrične koordinate unutar trokuta predstavljaju kombinaciju težina triju vrhova može se iskoristiti za interpolaciju svih atributa novog vrha. Funkcije `interpolate2D()` i `interpolate3D` čine upravo to. Uzimaju atribut od kontrolne točke i interpoliraju ga koristeći `gl_TessCoord`.

```
vec2 interpolate2D(vec2 v0, vec2 v1, vec2 v2)
{
    return vec2(gl_TessCoord.x) * v0 + vec2(gl_TessCoord.y) * v1 + vec2(
    gl_TessCoord.z) * v2;
}

vec3 interpolate3D(vec3 v0, vec3 v1, vec3 v2)
{
    return vec3(gl_TessCoord.x) * v0 + vec3(gl_TessCoord.y) * v1 + vec3(
    gl_TessCoord.z) * v2;
}
```

Isječak koda 4.5: Funkcije interpolacije

Uz navedene atribute, izlazni podatak sjenčara za evaluaciju teselacije je i pozicija Bézierove plohe. Izračunat će se uvrštavanjem baricentričnih koordinata plohe koja je ulazni podatak u jednadžbu Bézierovog trokuta.

```
outputPatchValue = oPatch.WorldPos_B300 * w * w * w +
                   outputPatch.WorldPos_B030 * u * u * u +
                   outputPatch.WorldPos_B003 * v * v * v +
                   outputPatch.WorldPos_B210 * 3.0 * w * w* u +
                   outputPatch.WorldPos_B120 * 3.0 * w * u * u +
                   outputPatch.WorldPos_B201 * 3.0 * w * w * v +
                   outputPatch.WorldPos_B021 * 3.0 * u * u * v +
                   outputPatch.WorldPos_B102 * 3.0 * w * v * v +
```

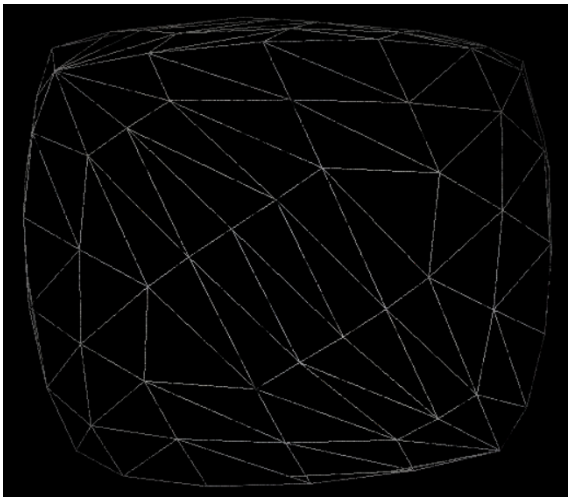
```
9                   outputPatch.WorldPos_B012 * 3.0 * u * v * v +
10                  outputPatch.WorldPos_B111 * 6.0 * w * u * v;
```

Isječak koda 4.6: Izračunavanje Bézierovog trokuta

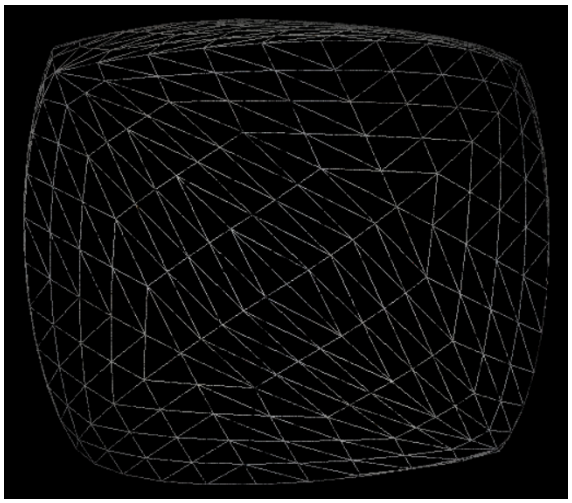
4.2 Prikaz rezultata

Slijedi prikaz rezultata dobivenih postupkom teselacije. Prvi teselirani objekt je jednostavna kocka, prikazana na slici (4.3). Rezultati se razlikuju ovisno o razini teselacije. Na slici (4.4) prikazan je teselirani objekt pri razini teselacije 5. Za teselaciju su u navedenom slučaju potrebna 444 trokuta. Slika (4.5) prikazuje teselaciju istog objekta na razini teselacije 9 i s 1800 generiranih trokuta. Najvjerniji prikaz stvarnog objekta daje teselacija na razini 50, pri čemu je generirano 46812 trokuta. Navedeni prikaz dan je na slici (4.6).

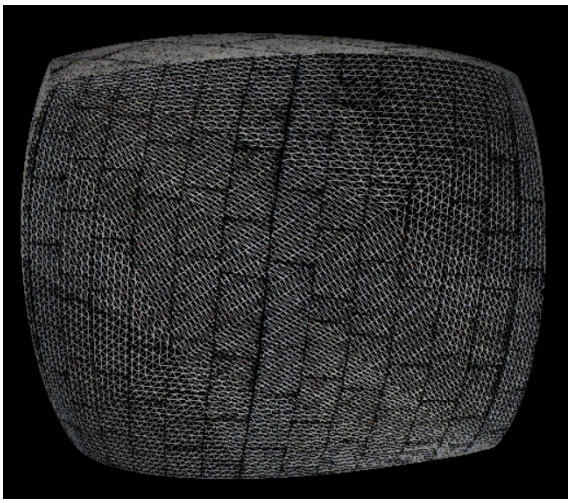
Slika 4.3



Slika 4.4



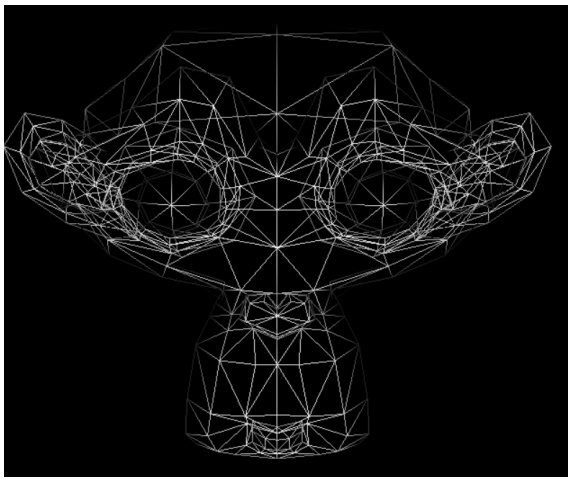
Slika 4.5



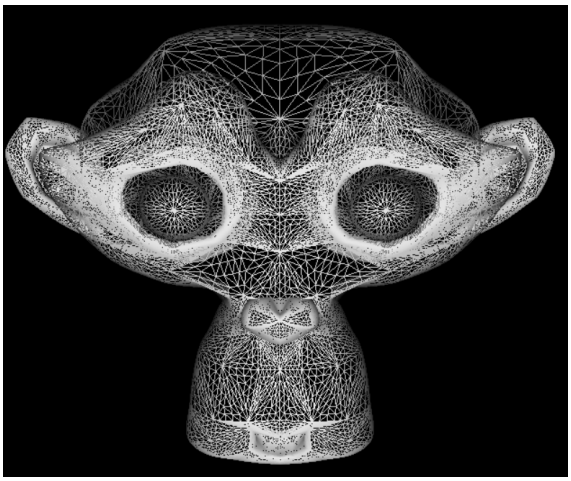
Slika 4.6

Sljedeći objekt na kojem je provedena teselacija prikazuje glavu majmuna (4.7). Slika (4.8) prikazuje teselaciju na razini 1, s 968 generiranih trokuta. Prikaz vrlo blizak originalnom objektu daje teselirani objekt s razinom teselacije 5 i 35816 generiranih trokuta. Prikazan je na slici (4.9).

Slika 4.7



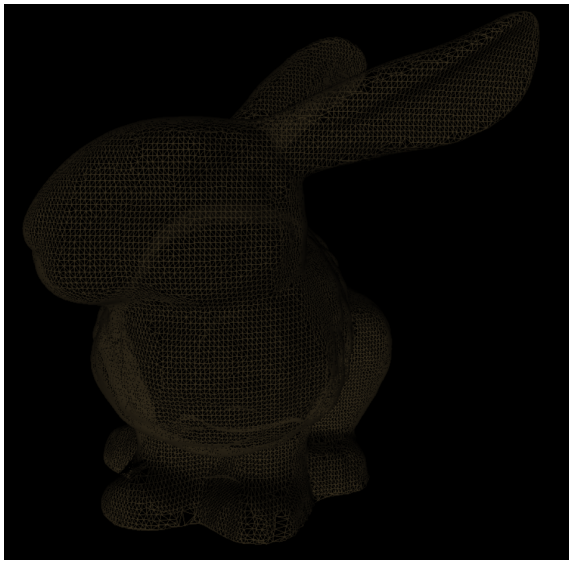
Slika 4.8



Slika 4.9

Posljednji objekt na kojem je provedena teselacija prikazuje zeca. Prikazan je na slici (4.10). Ovaj objekt je kompliciraniji od prethodnog i zadan je sa znatno većim brojem točaka. Već pri razini teselacije 1 generira 69666 trokuta i daje vrlo vjeran prikaz početnog objekta. Rezultat za navedene podatke je prikazan na slici (4.11).

Slika 4.10



Slika 4.11

Bibliografija

[1] *OGLdev Modern OpenGL Tutorials: Basic Tessellation*, http://ogldev.atspace.co.uk/www/tutorial30/tutorial30.html, posjećena siječanj, 2019.

[2] *OGLdev Modern OpenGL Tutorials: Triangle Tessellation*, http://ogldev.atspace.co.uk/www/tutorial31/tutorial31.html, posjećena siječanj, 2019.

[3] *OpenGL Wiki: Rendering Pipeline Overview*, https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview, posjećena siječanj, 2019.

[4] *OpenGL Wiki: Tessellation*, https://www.khronos.org/opengl/wiki/Tessellation, posjećena siječanj, 2019.

[5] *OpenGL*, https://www.opengl.org/, posjećena siječanj, 2019.

[6] Onur Rauf Bingol i Adarsh Krishnamurthy, *NURBS-Python: An open-source object-oriented NURBS modeling framework in Python*, SoftwareX (2019).

[7] F.J. Espino, M. Boo, M. Amor i J.D. Bruguera, *Adaptive Tessellation of NURBS Surfaces*, Journal of WSCG (2003).

[8] L. Piegl i W. Tiller, *The NURBS Book*, Springer, 1996.

Sažetak

U ovom radu prezentirani su različiti algoritmi teselacije NURBS ploha trokutima. Precizno su definirane plohe koje su potrebne za razumijevanje algoritama – Bézierova, B-splajn i NURBS ploha. Objašnjeno je da algoritmi teselacije NURBS ploha zahtijevaju prikaz NURBS plohe pomoću Bézierovih ploha. Objašnjeni su pojmovi uniformne, adaptivne i potpuno adaptivne teselacije te pripadnih testova za provjeru točnosti teselacije.
Nadalje, predstavljen je OpenGL, sučelje za programiranje grafičkih aplikacija. Opisana je grafička struktura teselacije u OpenGL-u. Poseban naglasak stavljen je na faze koje su odgovorne za teselaciju.
Definirana je razina teselacije te je objašnjena razlika između unutarnje i vanjske teselacije. Prikazano je kako razina teselacije djeluje na teselaciju pojedinog trokuta.
Za kraj, opisana je implementacija rješenja koje za teselaciju koristi poseban oblik Bézierove plohe – Bézierov trokut. Opisani su koraci algoritma te su dani rezultati za tri objekta. Prikazane su slike objekata prije i nakon teselacije, uz različite razine teselacije. Veća razina teselacije daje bolje rezultate.

Summary

This paper presents various triangle tessellation algorithms of NURBS surfaces. The surfaces required for understanding algorithms are precisely defined – the Bézier, B-spline and NURBS surface. It is explained that NURBS surface tessellation algorithms require the NURBS surface to be displayed using Bézier surfaces. The terms uniform, adaptive and fully adaptive tessellation are explained, as well as the corresponding tessellation accuracy tests.
Furthermore, OpenGL, the graphical applications programming interface is presented. The graphical tessellation structure in OpenGL is explained, and special emphasis is placed on the phases responsible for tessellation.
The level of tessellation is defined, and the difference between inner and outer tessellation is explained. The paper also demonstrates how the level of tessellation affects the tessellation of an individual triangle.
Finally, the implementation of a solution which uses a special form of the Bézier surface – the Bézier triangle is described. The algorithm steps are described, and the results for three objects are provided. Images of the object before and after the tessellation are provided, with various tessellation levels. The higher tessellation levels yield better results.

Životopis

Ana Mlinarić je rođena 28. lipnja 1995. godine u Zaboku. Pohađala je Osnovnu školu „Ljudevit Gaj“ Krapina, a nakon toga upisuje Opću gimnaziju u Srednjoj školi Krapina. Sudjeluje na školskim i županijskim natjecanjima iz matematike i hrvatskog jezika, a u prvom razredu srednje škole i na državnom natjecanju iz matematike. U četvrtom razredu polaže jezičnu diplomu za poznavanje njemačkog jezika “Deutsches Sprachdiplom der Kultusministerkonferenz – Zweite Stufe”. Gimnaziju završava s odličnim uspjehom i 2013. godine upisuje preddiplomski sveučilišni studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta. Preddiplomski studij završava 2016. godine te nakon toga upisuje diplomski sveučilišni studij Računarstva i matematike, također na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta. Na prvoj godini diplomskog studija prijavljuje se za sudjelovanje na ljetnom kampu kompanije Ericsson Nikola Tesla te sudjeluje na istom u srpnju i rujnu 2017. godine. U travnju 2018. zapošljava se u istoj kompaniji kao Junior Software Developer u odjelu istraživanja i razvoja. Tamo radi do rujna iste godine, kada prelazi u tvrtku Uprise, u kojoj je trenutno zaposlena kao student, na istoj poziciji Junior Software Developera.