Generativni model dubokog učenja GAN za povećanje rezolucije video sadržaja

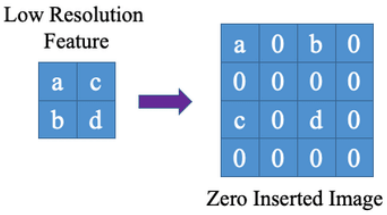
Dino Grgić
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave
dino.grgic@fer.hr

***Sažetak*—Prilikom izgradnje programa računalne grafike, često postoji potreba za povećanjem slike niske rezolucije (LR - *engl. low resolution)* u sliku visoke rezolucije (HR - *engl. high resolution)*. Jedan od takvih programa je poznati *Photoshop* koji u sebi implementira ovu funkcionalnost. Jedan od važnijih modela za primjene povećanje rezolucije slike je duboki generativni model GAN čiji trenirani modeli omogućavaju generiranje slika visoke rezolucije (GEN). Sljedeći korak u razvoju GAN arhitektura je generacija videa što je ništa drugo nego slijedni prikaz slika. Veliki izazov danas je optimizirati GAN kako bi povećavao rezoluciju videa u stvarnom vremenu.**

***Ključne riječi*—povećanje rezolucije, interpolacije, GAN, video sadržaj**

I. Uvod

Povećanje kvalitete rasterske slike postaje sve aktualniji problem u području računalne grafike i računalnog vida. Načini koji se rasterska slika može poboljšati su primjerice uklanjanje šuma (*engl. denoising*), restauracija te povećanje rezolucije bez gubitka kvalitete. U tu svrhu koristimo klasične metode ali i metode dubokog učenja te njihove modele. Problem kod povećanja rasterske slike je to što povećanjem slike dimenzije dobijemo veliki broj piksela s nepoznatim intenzitetom boje (Slika 1.).



Slika 1. Povećanje slike rezolucije 2×2 na rezoluciju 4×4 [1]

Iako postoji puno istraživanja u području slike, nedavno započinje trend i u povećanju kvalitete video sadržaja. Postoji veliki broj modela koji mogu jako lako povećati rezoluciju slike animiranoga sadržaja no problem nastaje kada pokušavamo raditi sa slikama iz stvarnog svijeta. Naime, slike nisu standardizirane te mogu biti različite kvalitete. Uz to postoje i razni problemi koji mogu pogoršati kvalitetu slike zbog načina rada kamere kojom su slike napravljene. U tu svrhu je 2022. kreiran set podataka s video sadržajem iz stvarnog svijeta koji je opisan u radu [2].

II. Metode

A. Klasične metode

Metode koje se dugi niz godina koriste u računalnoj grafici nazvat ćemo klasičnima. Većina ovih metoda zasniva se na interpolaciji. Interpolacija je matematička metoda koja omogućava predviđanje vrijednosti točke u rasponu poznatih vrijednosti. Svaka metoda ima svoje koristi, a kako se one koriste i u modernim metodama potrebno ih razumjeti. Neke od metoda za povećanje rezolucije slike su: metoda najbližih susjeda, bilinearna interpolacija, bikubična interpolacija itd. U svrhu izrade povećanje rezolucije video sadržaja kao najjednostavniji model engl. base model) koristit ćemo bikubičnu interpolaciju. Bikubična interpolacija nije intenzivna u performansama te daje najbolje rezultate od svih klasičnih metoda (Slika 2).

Slika 2. Kvaliteta metoda povećanja a) najbliži susjed, b) linearna interpolacija, c) bikubična interpolacija, d) originalna slika [3]

Bikubična interpolacija je metoda koja najbolje zadržava kvalitetu detalja i grubog oblika na slici. Za razliku od bilinearne interpolacije, bikubična interpolacija koristi 16 točaka za određivanje nepoznatog intenziteta piksela. Osim vrijednosti intenziteta u izračunu koristimo i derivacije u točkama. Za bilinearnu interpolaciju bilo nam je potrebno tri jednadžbe dok nam za bikubičnu interpolaciju treba 16 jednadžbi. Kako bismo pojednostavili proces koristit ćemo kubičnu konvolucijsku interpolaciju [4]. Metoda se svodi na računanje koeficijenata a_{ij} te izračun intenziteta koristeći jednadžbu:

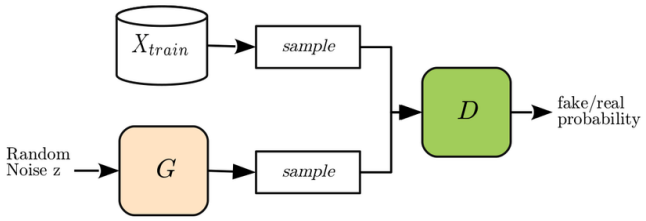
$$I(x,y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$$

B. Moderne metode

Metode koje koriste duboko učenje prilikom povećanje rezolucije nazvat ćemo moderne. Postoje različite vrste modela koje se mogu koristiti u svrhu povećanje rezolucije rasterske slike. U tu svrhu koriste se razni konvolucijski modeli te moderniji generativni suparnički modeli - GAN. Zadnjih pet godina nastao je veliki broj različitih GAN arhitektura u svrhu poboljšanja kvalitete slike. Jedna od tih arhitektura je ESRGAN treniran na podacima iz stvarnog svijeta.).

1) Arhitektura GAN: Dva modela koje GAN sadrži nazivamo generator (G) i diskriminator (D) (Slika 3). Treniranje GAN modela temelji se na *MinMax* igri između dvije mreže. Način na koji igra funkcionira je:

- Generativna mreža sliku niske rezolucije (LR) pretvara u sliku visoke rezolucije (HR) te stvara generiranu sliku (GEN)
- Diskriminativna mreža pokušava pogoditi koja slika je originalna: visoke rezolucije (HR) ili generirana slika (GEN)
- Cilj generativne mreže je prevariti diskriminativnu da je rezolucija slike točno povećana

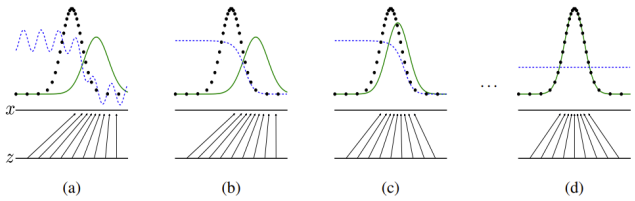


Slika 3. GAN arhitektura

2) Treniranja modela GAN: Cilj je da D mreža daje rezultate $\frac{1}{2}$ na cijelom setu podataka za treniranje (tj. da ne može razabrati originalnu od generirane slike). Obe mreže sastoje se od mnoštva perceptrona tako da je njihovo treniranje omogućeno unatražnim prolaskom *(engl. backpropagation)*. Težine trenirane mreže možemo pohraniti na disk i koristiti za povećanje rezolucija ostalih slika koje se ne nalaze u setu podataka. Svaki model dati će najbolje rezultate za svoj reprezentativni set podataka. Ovaj fenomen u strojnome učenju nazivamo *no free lunch*. Statistički gledano, model pokušava rekreirati podatak iz distribucije podataka na kojemu je učen. Primjerice, ako povećavamo rezoluciju slike prirode najbolje rezultate ćemo dobiti ako smo model trenirali na slikama prirode (Slika 4).

III. Skupovi podataka

Tipično skupove podataka u zadatku povećanje rezolucije video sadržaja razlikujemo na skupove s animiranim slikama i slikama iz stvarnog života (Slika 5). Modele za animirane skupova podataka je lakše trenirati za bolju kvalitetu HR slike. Razlog iza toga je što u animiranom sadržaju ne postoje nepravilnosti i nesavršenosti koje bi možda imali sa snimkom s video kamere. Generalno, skupovi podataka iz stvarnog svijeta



Slika 4. Distribucije prilikom treniranja modela GAN [5].

imaju više šuma te je teže dobro trenirati model koji će dobro generalizirati.

Slika 5. Različiti skupovi podataka za treniranje povećanje rezolucije slike

IV. Mjerila kvalitete

Postoje različita mjerila kojima možemo verificirati kvalitetu nastale slike. [1]

- Subjektivno mjerilo - Ispitati na velikom broju ispitanika kvalitetu videa s obzirom na referentni HR video
- LPIPS - *Learned Perceptual Image Patch Similarity* - Sličnost slika ispitana na pretreniranoj neuronskoj mreži. Tražen je manji rezultat.
- SSIM - *Structural similarity index measure*
- PSNR - *Peak signal-to-noise ratio*
- FPS - *Frames per second* - Za ocjenjivanje kvalitete generiranje videa u stvarnom vremenu

V. Nastavak rada

Sva navedena istraživanja koristit će se za izradu diplomskog rada. Zbog velikog broja radova i novih skupova podataka u posljednjih par godina možemo koristiti veliki broj mjerila i različitih skupova podatka. Rad se može nastaviti na optimizaciji modela za povećanje rezolucije u stvarnome vremenu

[1] https://videoprocessing.ai/benchmarks

Osim povećanje rezolucije slika, ove metode, ako su dovoljno brze, omogućavaju nam i povećanje rezolucije u stvarnome vremenu. Trenirani modeli mogu povećavati rezoluciju videa, vizualnih efekata u video igrama, prikaz simulacije fluida, itd. NVIDIA nudi svoje modele za povećanje rezolucije videoigara putem grafičkog toka čime se poboljšavaju performanse na starijima računalima[2].

Idući koraci na nastavak rada mogu biti sljedeći:

- Osmisliti manju arhitekturu s ciljem generiranja HR video sadržaja u stvarnom vremenu
- Ocijeniti kvalitetu modela s raznim mjerama dostupnim na https://videoprocessing.ai/benchmarks/video-super-resolution-participants.
- Pronaći primjenu na povećanje rezolucije tekstura i video sadržaja u video igrama

LITERATURA

[1] X. Zhang, S. Karaman, and S. Chang, "Detecting and simulating artifacts in gan fake images," 07 2019.

[2] H. Yue, Z. Zhang, and J. Yang, "Real-rawvsr: Real-world raw video super-resolution with a benchmark dataset," in *European Conference on Computer Vision*. Springer, 2022, pp. 608–624.

[3] J. Xu, Z. Chang, J.-L. Fan, X. Zhao, X. Wu, Y. Wang, and X. Zhang, "Super-resolution via adaptive combination of color channels," *Multimedia Tools and Applications*, vol. 76, 01 2017.

[4] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[2]https://www.nvidia.com/en-us/geforce/news/nvidia-image-scaler-dlss-rtx-november-2021-updates/

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Vizualizacija najkraće krivulje između dvije točke na modelu visinske mape

Josip Komljenović

Voditelj: *Željka Mihajlović*

Zagreb, svibanj, 2022.

Sadržaj

1. Uvod

Pronalaženje najkraćeg puta između dvije točke na modelu visinske mape je jedan od čestih optimizacijskih problema. Svoju primjenu može naći u raznim ljudskim djelatnostima, od prostornog planiranja do modeliranja kretanja objekata. Ovaj rad obrađuje problematiku traženja pravog algoritma za taj zadatak, moguća ubrzanja algoritma kao i razna ograničenja koja se mogu postaviti ovisno o karakteristikama krivulje koje želimo ostvariti. Rad detaljno prikazuje teoretsku podlogu jednog od algoritama za izvršavanje ovog zadatka te implementaciju tog algoritma u programskom okruženju *Unreal Engine* koristeći programski jezik C++. Prikazana su vremenska i memorijska svojstva algoritma kao i mogući prijedlozi za poboljšanja performansi algoritma

2. Seminarski rad

2.1. Algoritmi za traženje najkraćeg puta

Budući da je traženje najkraćeg puta na modelu visinske mape optimizacijski problem za njega je razvijeno mnoštvo različitih algoritama. Prvi pokušaji su bili usmjereni u primjenu klasičnih algoritama obilaska grafa, poput Dijkstrinog algoritma ili algoritma A*. Dok prethodno navedeni algoritmi ostvaruju odlične rezultate na dvodimenzionalnim površinama, rezultati koje ostvaruju ti algoritmi u trodimenzionalnom prostoru nisu najbolji. Specifično, kod algoritma A* veliki problem predstavlja odrediti pravu funkciju heuristike za preostalu udaljenost do cilja. Standardne funkcije heuristike, poput izračuna Manhattan udaljenosti ili zračne udaljenosti nisu najbolje rješenje za trodimenzionalni prostor. Zbog karakteristika prostora u svakoj točki duž puta na visinskoj mapi bi, osim dvodimenzionalne udaljenosti, trebalo u obzir uzeti i nagib terena kako bismo dobili preciznu procjenu udaljenosti, što je složeni izračun. Također na vremensku i prostornu složenost algoritma A* velik utjecaj imaju duljina puta i karakteristike površine visinske mape. U slučaju da je konačni put dugačak i pun zavoja program bi trebao u istom trenutku u memoriji držati relativno veliku sortiranu listu, što ovisno o implementaciji može dovesti do loših performansi. Zbog prethodno navedenih razloga fokus je prebačen na otkrivanje drugih algoritama koji će koristiti osnovna svojstva algoritma A*, ali uz određene promjene specifične za ovaj problem. Vrijedi izdvojiti dva algoritma. Prvi algoritam je algoritam neizrazite transformacije udaljenosti korištenjem tehnika dinamičkog programiranja. Algoritam definira duljinu puta na neizrazitom podskupu i zatim traži infimum duljina svih puteva između dvije točke. Algoritam generira vrlo kvalitetne putove, međutim nema dobre performanse i ima složenu implementaciju. Drugi algoritam je algoritam transformacije udaljenosti na zakrivljenom prostoru. Algoritam je prvotno osmišljen za analizu slika koje sadrže razne nijanse sive boje, međutim, budući da se visinska mapa vrlo često i prikazuje kao takva slika, ovaj algoritam je našao svoju primjenu i na ovom problemu. Zbog svojih performansi i jednostavnosti implementacije, u nastavku ovog rada će detaljno biti opisan upravo ovaj algoritam.

2.2. Algoritam transformacije udaljenosti na zakrivljenom prostoru

Algoritam transformacije udaljenosti kao ulaz prima sliku ispunjenu nijansom sive boje. Tu sliku možemo shvatiti kao dvodimenzionalnu listu kod koje svaki piksel ima svoju poziciju (stupac i rekad predstavljaju x i y koordinatu u trodimenzionalnom prostoru), a vrijednost nijanse sive boje predstavlja visinu, odnosno z koordinatu u trodimenzionalnom prostoru. Svijetlije (bijele) nijanse se interpretiraju kao područja veće visine, a tamnije (crne) nijanse kao područja manje visine. U nastavku će slika visinske mape biti označena slovom G. Algoritam osim slike visinske mape također koristi i pomoćnu sliku F. F „je jednake veličine kao i G i predstavlja sliku udaljenosti. Nakon što smo definirali sliku potrebno je definirati prostor za računanje, kojeg označimo s X, i pozadinski prostor kojeg označimo s X^C. Svaki piksel slike F mora pripadati ili prostoru X, ili prostoru X^C, a presjek ta dva prostora mora biti prazan skup. Kardinalitet skupa X je veći od kardinaliteta skupa X^C. Potrebno je također definirati i susjedstvo dva piksela. Svaki piksel ima maksimalno 8 susjeda (po jednog gore, dolje, lijevo i desno, te susjede u dijagonalnim smjerovima). Udaljenost između dva piksela označimo s $d(\overrightarrow{p_i}, \overrightarrow{p_{i+1}})$. Postoje dvije varijante za izračun udaljenosti između dva piksela. Prva varijanta je koristeći formulu $d(\overrightarrow{p_i}, \overrightarrow{p_{i+1}}) = |G(p_i) - G(\overrightarrow{p_{i+1}})| + 1$. Prednost ove varijante je dobra brzina izvođenja, međutim nedostatak je što se dijagonalne susjede smatra jednako udaljenima kao i neposredne susjede. Složenija, ali i preciznija varijanta je transformacija težinskih udaljenosti. Ta formula glasi $d(\overrightarrow{p_i}, \overrightarrow{p_{i+1}}) = \sqrt{|G(p_i) - G(\overrightarrow{p_{i+1}})|^2 + k}$, gdje k iznosi 1 ukoliko je riječ o neposrednom susjedu, a 2 ukoliko je riječ o dijagonalnom susjedu. Na početku algoritma vrijednost svih piksela na slici F koji pripadaju skupu X postavimo na $+\infty$ (odnosno u implementaciji najveću vrijednost koju piksel može poprimiti), a svim pikselima unutar skupa X^C dodijelimo vrijednost 0. Nakon što smo definirali sve potrebno za algoritam krećemo u izvođenje algoritma. Algoritam se provodi u najmanje dvije iteracije. U prvoj iteraciji algoritma počinjemo od gornjeg lijevog kuta i krećemo se prema desno i dolje. Za svaki piksel računamo novu vrijednost piksela na sljedeći način: $F_{i+1}(\overrightarrow{p_c}) = \min\left(F_i(\overrightarrow{p_c}), \min_{\vec{p} \in M_1}(d(\overrightarrow{p_c}, \vec{p}) + F(\vec{p}))\right)$, gdje je M_1 skup u kojem se nalaze sjeverni, zapadni, sjeverozapadni i sjeveroistočni susjed piksela $\overrightarrow{p_c}$. Kada je algoritam

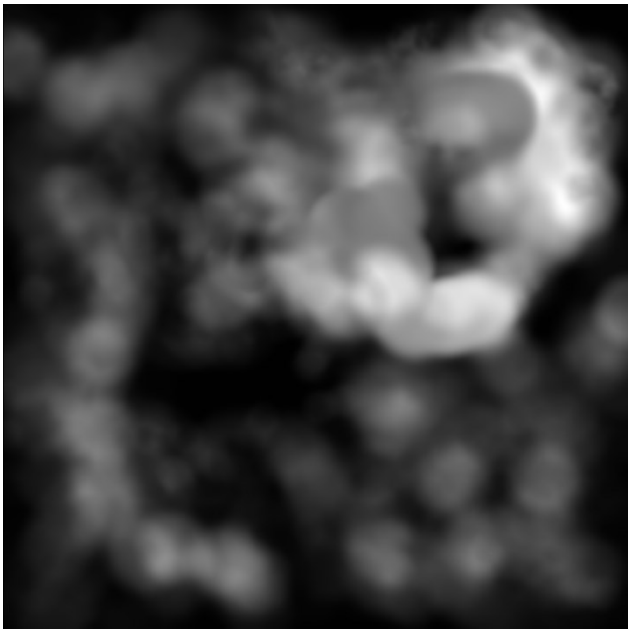
došao do donjeg desnog kuta kreće se s drugom iteracijom. U drugoj iteraciji se kreće od donjeg desnog kuta i ide lijevo gore. Provodi se isti izračun, uz jedinu razliku što se skup M_1 zamjenjuje skupom M_2 u kojem se nalaze južni, istočni, jugoistočni i jugozapadni susjed. Druga iteracija završava u gornjem lijevom kutu nakon čega je algoritam generirao sliku udaljenosti.

Prethodno navedenim korakom je završio algoritam transformacije udaljenosti na zakrivljenom prostoru. Nakon toga je potrebno iskoristiti taj algoritam za pronalazak najkraćeg puta. Za put zadajemo dvije točke, $\overrightarrow{p_S}$ koja predstavlja početnu točku puta, i $\overrightarrow{p_E}$ koja predstavlja završnu točku puta. Algoritam traženja najkraćeg puta započinjemo generiranjem udaljenosti za početnu točku F_S. Algoritmu transformacije udaljenosti kao skup X^C predajemo samo piksel $\overrightarrow{p_S}$, dok svi ostali pikseli pripadaju skupu X. Nakon što smo generirali sliku udaljenosti za početnu točku generiramo i sliku udaljenosti za završnu točku F_E (ovog puta skup X^C čini $\overrightarrow{p_E}$). Nakon što smo generirali F_S i F_E generiramo i sliku D_R. Za sliku D_R vrijedi sljedeće: svaki piksel u slici D_R je jednak zbroju piksela na istoj poziciji slike F_S i F_E, odnosno $D_R(\vec{p}) = F_S(\vec{p}) + F_E(\vec{p})$. Slika D_R je konačna slika koja nam služi za generiranje puta. Put čine točke s minimalnom vrijednošću na slici D_R. Do minimalne vrijednosti najlakše dolazimo na način da uzmemo vrijednost D_R za početnu ili završnu točku (jer put mora prolaziti kroz njih). Sve ostale točke na slici će imati ili veću vrijednost (ne čine najkraći put) ili jednaku vrijednost (čine najkraći put).

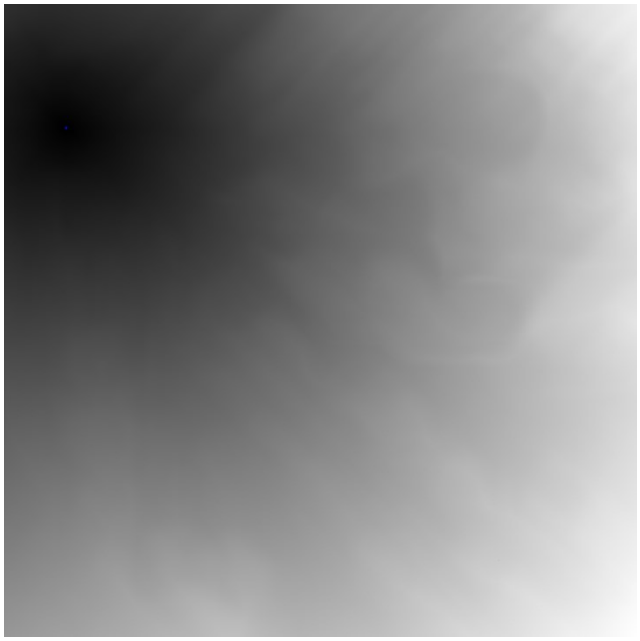
Zbog svojih svojstava ovaj algoritam može imati primjenu i u složenijim slučajevima od traženja najkraćeg puta između dvije točke. Algoritam se može primijeniti za traženje najkraćeg puta između bilo koja 2 skupa točaka. Primjerice, u situaciji u kojoj imamo 3 grada: A, B i C, i želimo napraviti cestu koja spaja grad C sa gradom A ili B uz uvjet da cesta bude najkraća sve što je potrebno napraviti je za sliku F_E u skup X^C umjesto jedne točke dodati dvije točke ($\overrightarrow{p_A}$ i $\overrightarrow{p_B}$) i algoritam će generirati najkraći put između grada C i jednog od preostalih gradova. U skupu X^C se može nalaziti proizvoljan broj točaka.

2.3. Primjer izvođenja algoritma

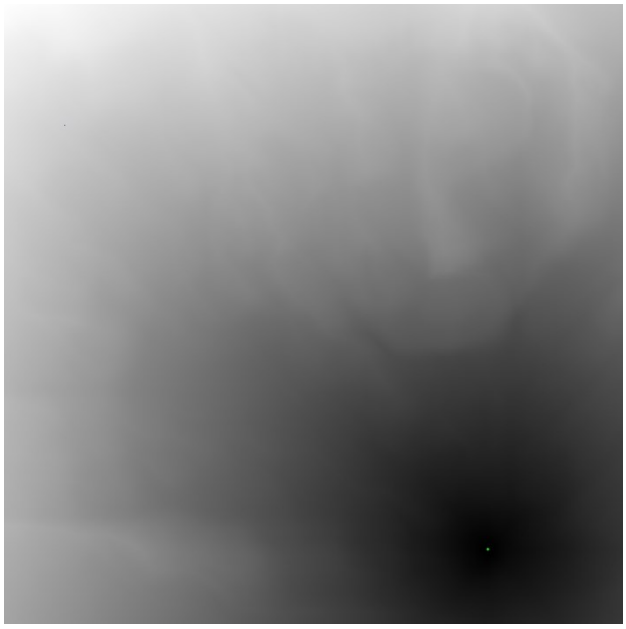
Sljedeći primjer prikazuje korake algoritma na ručno generiranoj visinskoj mapi. Slike su generirane za polja veličine 1024x1024. konkretna vrijednost svakog piksela je skalirana na interval $[0, 255]$ kako bi se slika lakše prikazala pomoću nijansi crne boje. Crna boja predstavlja vrijednost 0, a bijela vrijednost 255. Na završnoj slici D_Ržutom bojom je prikazan najkraći put između dvije točke.



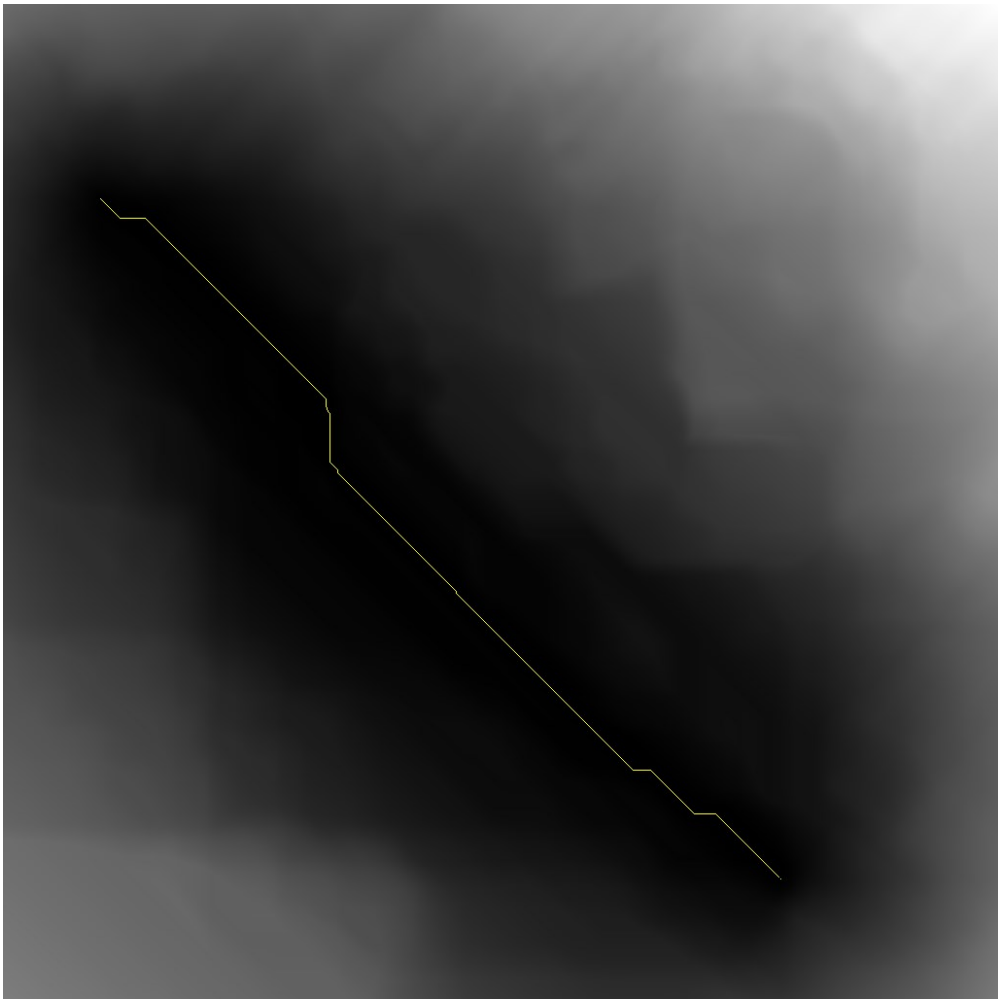
Slika 2-1 Visinska mapa



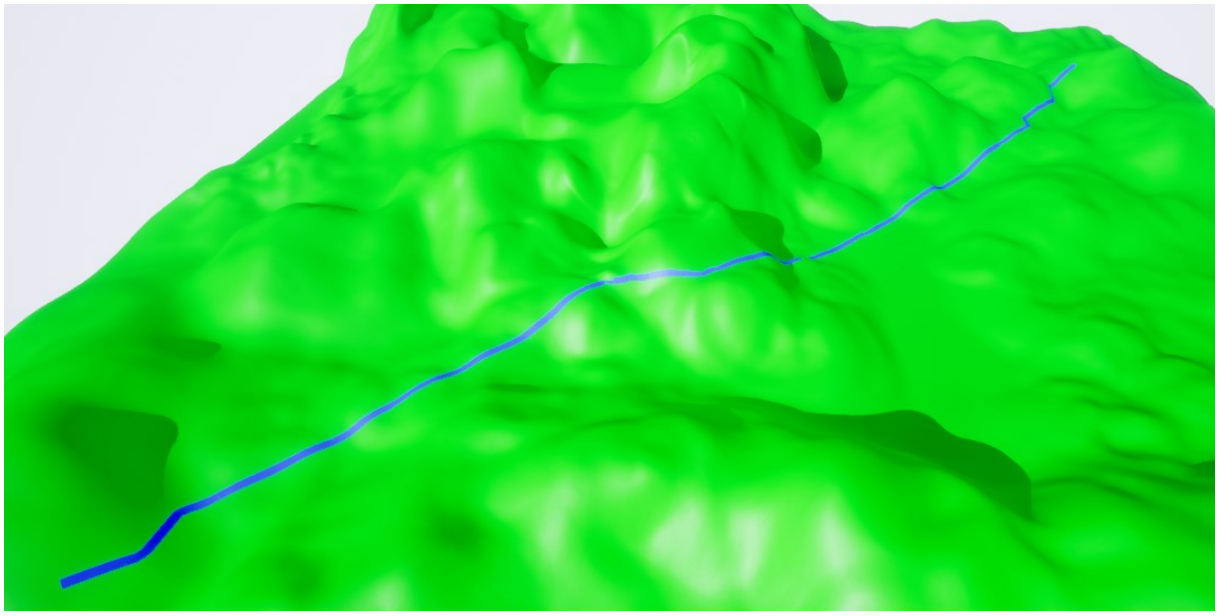
Slika 2-2 Slika udaljenosti za početnu točku



Slika 2-3 Slika udaljenosti za završnu točku



Slika 2-4 Konačan put



Slika 2-5 Vizualizacija krivulje na visinskoj mapi

2.4. Implementacija

Tema ovog rada je implementirana u programskom okruženju *Unreal Engine* koristeći programski jezik C++. Visinska mapa se može generirati koristeći ugrađene alate *Unreal Engine*a, ili ju je moguće učitati iz datoteke. Nakon što je visinska mapa postavljena potrebno je doći do vrijednosti visina na mapi. Za to je korišten Line Trace algoritam čija implementacija dolazi s instalacijom sustava. Generira se linija unutar intervala minimalne i maksimalne visine mape i računa točka sudara sa visinskom mapom te se uzima z koordinata te točke. Nakon toga se stvara polje realnih brojeva koje će predstavljati sliku udaljenosti za početnu točku. Kreiranje i inicijalizacija polja su provedene sljedećim programskim odsječkom:

```
float* StartImage = new float[imageSize * imageSize];
for (int32 y = 0; y < imageSize; y++) {
	for (int32 x = 0; x < imageSize; x++) {
		int32 PixelIndex = ((y * imageSize) + x);
		if (x != startLocation.X || y != startLocation.Y)
			StartImage[PixelIndex] = FLT_MAX;
		else StartImage[PixelIndex] = 0;
	}
}
```

Nakon toga Algoritam kreće u prvu iteraciju. Za svaki piksel slike se izvršava sljedeći kod:

```
for (int32 y = 0; y < imageSize; y++) {
	for (int32 x = 0; x < imageSize; x++) {
		int32 PixelIndex = ((y * imageSize) + x);
		float newValue = StartImage[PixelIndex];
		float selfValue = OriginalImage[PixelIndex];
		float northWest = FLT_MAX; float north = FLT_MAX; float northEast = 
FLT_MAX; float west = FLT_MAX;
		if (x > 0 && y > 0) northWest = abs(OriginalImage[(y - 1) * imageSize + 
(x - 1)] - selfValue) + UE_SQRT_2 * PIXEL_GAP + StartImage[(y - 1) * imageSize + (x - 
1)];
		if (y > 0) north = abs(OriginalImage[(y - 1) * imageSize + x] - 
selfValue) + 111 + StartImage[(y - 1) * imageSize + x];
		if (y > 0 && x < imageSize - 1)  northEast = abs(OriginalImage[(y - 1) * 
imageSize + (x + 1)] - selfValue) + UE_SQRT_2 * PIXEL_GAP + StartImage[(y - 1) * 
imageSize + (x + 1)];
		if (x > 0) west = abs(OriginalImage[y * imageSize + (x - 1)] - selfValue) 
+ PIXEL_GAP + StartImage[y * imageSize + (x - 1)];
		StartImage[curPixelIndex] = FMath::Min(newValue, FMath::Min(northWest, 
FMath::Min(north, FMath::Min(northEast, west))));
	}
}
```

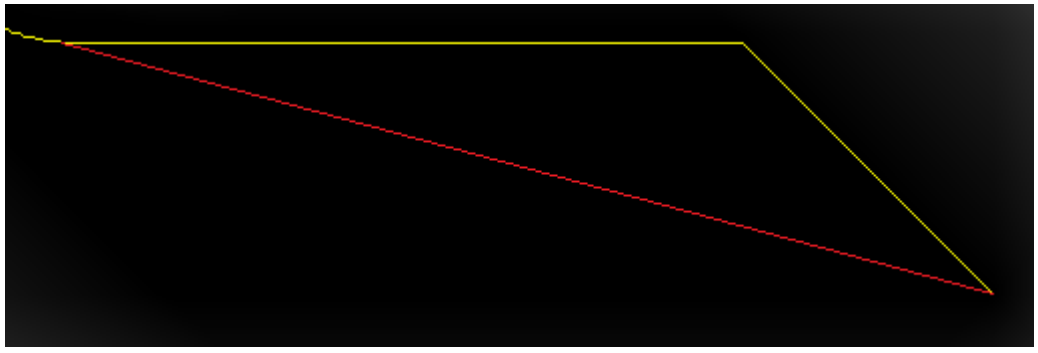
Sljedeći korak je napraviti drugi prolazak kroz sliku, uz promjenu redoslijeda piksela i susjednih piksela koji se promatraju na način koji je detaljno opisan u dijelu 2.2. Istim postupkom se generira i slika udaljenosti za završnu točku puta. Potrebno je još samo generirati konačnu sliku tako da zbrojimo vrijednosti te dvije slike udaljenosti. Nakon toga preostaje još samo odrediti točke koje pripadaju putu. To se može učiniti jednostavnim prolaskom od početne točke tražeći najmanju susjednu vrijednost sve do završne točke. Time smo dobili povezane točke puta i na temelju njih se na kraju iscrtava krivulja. Za vizualizaciju generiranih slika koristi se sljedeća pomoćna funkcija:

```
uint8* Pixels = new uint8[1024 * 1024 * 4];
float min = FLT_MAX, max = FLT_MIN;
for (int32 y = 0; y < imageSize; y++) {
	for (int32 x = 0; x < imageSize; x++) {
		int32 curPixelIndex = ((y * imageSize) + x);
		if (image[curPixelIndex] < min) min = image[curPixelIndex];
		if (image[curPixelIndex] > max) max = image[curPixelIndex];
	}
}
for (int32 y = 0; y < imageSize; y++) {
	for (int32 x = 0; x < imageSize; x++) {
		int32 curPixelIndex = ((y * imageSize) + x);
		uint8 new_value = 255 * ((image[curPixelIndex] - min) / (max - min));
		Pixels[4 * curPixelIndex] = new_value;
		Pixels[4 * curPixelIndex + 1] = new_value;
		Pixels[4 * curPixelIndex + 2] = new_value;
		}
	}
}
```

2.5. Analiza vremenske složenosti i dodatna poboljšanja algoritma

Ova implementacija je testirana na raznim visinskim mapama. Veličine mapa u koordinatnom sustavu *Unreal Enginea* iznose 114200x114200. Trajanje izračuna najkraće krivulje i njenog iscrtavanja traje nekoliko sekundi. Zbog svojih svojstava, ovaj algoritam u osnovnoj verziji ima približno jednaku vremensku i prostornu složenost za najbolji i najgori slučaj, što ga čini dobrim za algoritam koji radi u stvarnom vremenu. Ukoliko označimo veličinu mape kao $n \times n$ tada je vremenska i prostorna složenost algoritma $O(n^2)$. Ovaj algoritam se može vrlo dobro paralelizirati. Primjerice. Računanje slike udaljenosti za početnu i za završnu točku su dvije potpuno nezavisne radnje, te se one mogu izvoditi istovremeno. Također se istodobno može izvoditi izračunavanje vrijednosti udaljenosti piksela od svakog njegovog susjeda.

Postoje i dodatna poboljšanja koja bi algoritam mogao poprimiti. U trenutnoj verziji budući da algoritam za sliku udaljenosti provjerava 8 susjeda to znači da se krivulja može generirati u 8 smjerova, međutim to nije dovoljno kako bi se generirao optimalan put. Problem najviše dolazi do izražaja u dijelovima mape sa jednakom visinom. Sljedeća slika opisuje ovaj problem



Na slici je žutom bojom označen generirani put, a crvenom optimalni put. Za sustav ta dva puta imaju jednaku duljinu, međutim crveni put je kraći. Ovaj problem možemo riješiti tako da u zadnjem koraku nakon što generiramo put prođemo kroz zavoje na putu i gledamo zračne udaljenosti tri susjedna zavoja. Ako je njihova udaljenost velika generiramo privremeni put koji spaja prvu i treću točku. Ukoliko je taj put kraći od početnog obrišemo drugu točku iz puta. Ukoliko je put dulji to znači da je potreban zavoj na drugoj točki te se ništa ne mijenja.

3. Zaključak

U ovom radu je predstavljen jedan od pristupa pronalaska optimalnog puta između dvije točke na modelu visinske mape. Algoritam ne daje garanciju da je pronađen najkraći put, međutim na jednostavan način i relativno brzo vrijeme daje dovoljno dobar put, koji se uz male nadogradnje može dodatno poboljšati. Ovo područje je i dalje otvoreno za istraživanje, stoga se možda u budućnosti otkrije još bolji algoritam od prethodno prikazanog.

4. Sažetak

Ovaj rad obrađuje tematiku traženja najkraće krivulje između dvije točke na trodimenzijskoj površini i prikazivanje te krivulje. Rad navodi nekoliko algoritama koji se mogu koristiti za traženje najkraće udaljenosti u trodimenzijskom prostoru, uz detaljno objašnjenje algoritma transformacije udaljenosti na zakrivljenom prostoru. Rad također prikazuje implementaciju algoritma u Unreal Engine okruženju, analizira performanse ovisno o parametrima i navodi moguće prilagodbe.

5. Literatura

- Ikonen L., Toivanen P. Shortest Route on Height Map Using Gray-Level Distance Transforms
- Saha P. K., Wehrli F. W., Gomberg B. R. Fuzzy Distance Transform: Theory, Algorithms and Applications
- Toivanen, P. New geodesic distance transforms for gray-scale images
- Wichmann D. R., Wuensche B. Automated route finding on digital terrains

Vizualizacija glazbe

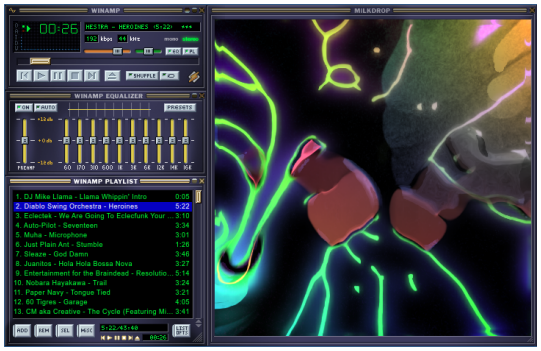
Ivan Vlahov
mentorica Željka Mihajlović, prof. dr. sc.
Fakultet elektrotehnike i računarstva, Zagreb
iv51747@fer.hr

***Sažetak*—U ovom radu prikazano je više metoda vizualizacije glazbe. Prikazuje se jednostavna vizualizacija amplitudom (glasnoćom), rastav zvučnih signala korištenjem algoritma brze Fourierove transformacije (FFT), te se daje pregled dostupne literature o vizualizaciji glazbe korištenjem generativnih suparničkih mreža (GAN). Prikazana je implementacija vizualizacije amplitudom i rastavom signala FFT-om u programskom paketu Processing.**

***Ključne riječi*—vizualizacija, glazba, video, FFT, GAN**

I. Uvod

Vizualizacija glazbe je relativno nova grana umjetnosti koja se pojavila sedamdesetih godina prošlog stoljeća. Popularnost je stekla petanestak godina kasnije, kada su se počeli pojavljivati programi za vizualizaciju glazbe za osobna računala. [3] Danas se vizualizacija glazbe koristi za razne stvari, od izrade vizualnih efekata na koncertima, glazbenih video spotova, pa sve do ukrasnih svjećica na božićnim drvcima. Neki od programa koji se koriste za vizualizaciju su Renderforest, Synesthesia i Videobolt.



Slika 1. Program za reprodukciju glazbe Winamp, Izvor: [5]

Iako već postoji puno programa za vizualizaciju, u ovom radu će biti predstavljene tri metode vizualizacije: vizualizacija amplitudom (glasnoćom) zvučnog signala, vizualizacija FFT-om, odnosno rastavom zvučnih valova na sinusoidne signale različitih frekvencija i amplituda, te se na kraju daje kratak uvod u generativne suparničke mreže (eng. *generative adversarial networks, GANs*) i uvid u dva rada u kojem se glazba vizualizira korištenjem GAN-ova.

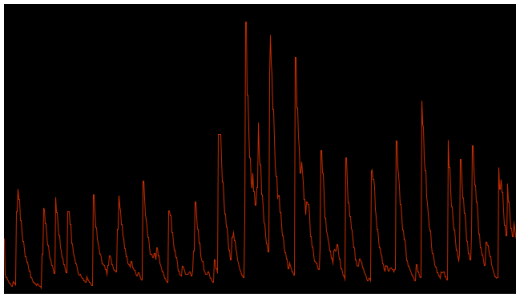
Za prve dvije metode dane su i snimke zaslona implementacije napravljene u programskom paketu Processing korištenjem biblioteke *Sound*. [1]

II. Jednostavne metode vizualizacije

A. Vizualizacija amplitudom

Prva tehnika je vrlo jednostavna - varijabla koja se koristi za animaciju objekata jest amplituda zvučnog signala u određenom trenutku. Korištenje ove metode ne omogućuje veliku fleksibilnost, jer svi objekti koje animiramo ovise o istoj varijabli.

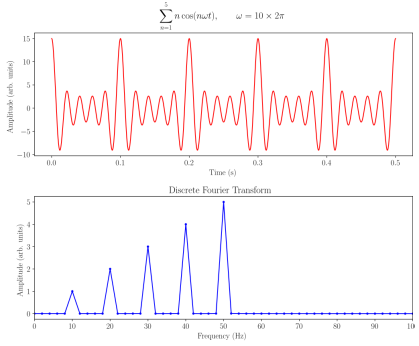
U programskom paketu Processing, ovakvu vizualizaciju možemo izvesti korištenjem razreda `Amplitude`. Na Slici 2 prikazana je snimka zaslona vizualizacije amplitude (y-os) u odnosu na vrijeme (x-os).



Slika 2. Vizualizacija glasnoćom u Processingu

B. Vizualizacija FFT-om

Druga tehnika je složenija, a radi se o rastavu zvučnih valova na zbroj sinusoida korištenjem algoritma brze Fourierove transformacije (eng. *Fast Fourier Transform, FFT*). Ovaj algoritam pretvara jedan uzorak zvučnog zapisa iz vremenske domene u frekvencijsku.



Slika 3. Primjer rastava korištenjem FFT-a, Izvor: [4]

Ovaj pristup omogućava puno veću prilagodbu, jer se objekti na zaslonu mogu animirati npr. ovisno o basu, vokalima i sl., tako što se filtriraju određene frekvencije, te se one postave kao argumenti funkcija kojima animiramo.

U *Sound* biblioteci programskog paketa Processing, ovaj algoritam dolazi ugrađen u razred `FFT`, te se na Slici 4 može vidjeti snimka zaslona implementacije jednostavnog programa za vizualizaciju, u kojem je prikazana slika, naslov i autor određene pjesme, a ispod tih informacija se nalazi stupčasti grafikon koji prikazuje koliko je u tom trenutku velika amplituda određenih frekvencija koje sačinjavaju uzorak zvučnih signala.

Slika 4. Vizualizacija FFT-om u Processingu

III. Vizualizacija GAN-om

Generativne suparničke mreže, ili kraće GAN-ovi, su modeli neuronskih mreža koji se sastoje od dva dijela: prvi dio je generator, koji stvara izlazne podatke pokušavajući ih učiniti što sličnijima podacima iz skupa za učenje, a drugi dio je diskriminator, koji za određene podatke pokušava prepoznati radi li se o pravim podacima iz skupa za učenje ili o generiranim podacima. Ova dva modela djeluju kao suparnici - generator pokušava "prevariti" diskriminatorsku mrežu, a diskriminator pokušava "otkriti" razlike između stvarnih i generiranih podataka.

GAN-ovi su relativno nova metoda, objavljeni u radu Iana Goodfellowa iz 2014. godine [12]. Na temelju tog rada, kroz sljedećih nekoliko godina pojavili su se modeli koji generiraju jako realistične slike ljudi, životinja, umjetničkih djela itd. Primjer takvog modela je StyleGAN iz 2019., koji se može i isprobati u aplikaciji This person does not exist [6].

Slika 5. Slike koje je generirao StyleGAN; ni čovjek ni mačka na slici ne postoje. Izvor: [6] [7]

A. Vizualizacija StyleGAN-om

U radu Stylizing Audio Reactive Visuals [8], autori predlažu metodu vizualizacije glazbe korištenjem GAN-ova, konkretno modela StyleGAN. Ideja rada jest da se vizualizacija provodi u tri dijela: izdvajanje značajki, šetnja po ulaznom prostoru i mapiranje značajki u StyleGAN.

Značajke se izdvajaju tako što se dijele na niske, srednje i visoke pojaseve korištenjem audio-filtera. Za svaki uzorak zvučnog signala računa se razlika ovih triju značajki u odnosu na prošli uzorak te se pomoću njih provodi šetnja po ulaznom prostoru. Za svaku značajku frekvencije se generira slučajni vektor čije dimenzije odgovaraju dimenzijama ulaznog prostora, a magnituda mu se postavlja na magnitudu odgovarajuće značajke frekvencije. Zatim se ti novogenerirani vektori dodaju ulaznim vektorima iz prošlog koraka. Ovaj pristup omogućava gladak prijelaz između slika.

Slika 6. Primjer nekoliko slika iz vizualizacije StyleGAN-om, Izvor: [8]

B. Vizualizacija BigGAN-om

Drugi primjer modela korištenog za vizualizaciju glazbe je BigGAN, u radu The Deep Music Visualizer [9]. Ulazni podaci ovog modela sastoje se od vektora s 1000 stupaca koji odgovara jednom od 1000 ImageNet razreda različitih objekata [10], te vektora šuma sa 128 stupaca pomoću kojeg se postavljaju različite izlazne značajke poput boje, dimenzije objekata, položaja, orijentacije i sl.

Autor ovog rada također provodi izdvajanje značajki i mapiranje na GAN. Na početku se odabire do 12 različitih ImageNet razreda za koje se želi generirati vizualizacija. Visina tona je prva izdvojena značajka, kojom se upravlja koji će se od 12 razreda objekata prikazati. S druge strane, tempo i glasnoća upravljaju vektorom šuma. Slično kao i u prethodnom primjeru, ovaj pristup omogućava gladak prijelaz između slika.

Slika 7. Primjer nekoliko slika iz vizualizacije BigGAN-om, Izvor: [13]

IV. Diskusija, daljnji rad i zaključak

Vizualizacija glazbe je zanimljivo, još uvijek novo područje u kojem i dalje ima sadržaja za istraživanje. S obzirom na svoj status u popularnoj kulturi i svakodnevnom životu, za očekivati je da će se i dalje razvijati, i da ćemo sve češće viđati glazbene spotove koje je u potpunosti generirao model umjetne inteligencije.

Ovaj rad pokriva samo uvod u jednostavne metode vizualizacije, kao i jednostavne implementacije navedenih metoda. Kombiniranjem ovih metoda, korištenjem filtera različitih frekvencijskih pojaseva i daljnjim istraživanjem različitih generativnih modela, ostavljen je velik prostor za daljnje istraživanje metoda vizualizacije glazbe.

Programski kod korišten u ovom radu može se pronaći na GitHub stranici autora [11].

Literatura

[1] Processing Sound Library,
https://processing.org/reference/libraries/sound/index.html
[2] Let's learn about waveforms
https://pudding.cool/2018/02/waveforms/
[3] The basics of Music Visualizers
https://www.ipr.edu/blogs/audio-production/
the-basics-of-music-visualizers/
[4] Fast Fourier Transform
https://en.wikipedia.org/wiki/Fast_Fourier_transform
[5] Webamp
https://webamp.org/
[6] This person does not exist
https://www.thispersondoesnotexist.com/
[7] This cat does not exist
https://thiscatdoesnotexist.com/
[8] Stylizing Audio Reactive Visuals, Han-Hung Lee et al.
https://neurips2019creativity.github.io/doc/Stylizing%20Audio%
20Reactive%20Visuals.pdf
[9] The Deep Music Visualizer, Matt Siegelman
https://github.com/msieg/deep-music-visualizer
[10] IMAGENET 1000 Class List
https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/
IMAGENET/
[11] music-visualisation, I. Vlahov
https://github.com/vlahovivan/music-visualisation
[12] Generative Adversarial Nets, I. Goodfellow et al.
https://arxiv.org/pdf/1406.2661.pdf
[13] Deep Music Visualizer: Where is my mind - Maxence Cyrin
https://vimeo.com/365200802?signup=true

Krpanje rupa u trodimenzionalnim poligonalnim mrežama

B. Cafuk*
*Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
borna.cafuk@fer.hr

***Sažetak*—Rad daje uvod u načine kako mreže s rupama nastaju i zašto su takve mreže problematične za korištenje u različitim procesima u računalnoj grafici. Nakon toga rad daje pregled postojeće literature koja se bavi tematikom krpanja rupa u trodimenzionalnim mrežama, kao i grubi pregled u njoj opisanih algoritama za popravak mreža s rupama. Rad uspoređuje u literaturi opisane pristupe na temelju uvjeta potrebnih za rad opisanih rješenja.**

***Ključne riječi*—računalna grafika, poligonalne mreže, topologija**

I. Uvod

Trodimenzionalni predmeti se u računalnoj grafici najčešće modeliraju mrežama poligona. Objekti koji se modeliraju su često puni i modelira se samo izgled njihove površine, te takav model ne bi trebao u sebi imati praznine. Nekoliko razloga zašto su praznine u poligonalnim mrežama nepoželjne su:

- kod neprozirnih objekata se vidi da su šuplji i da je modelirana samo njihova površinai kroz prazninu se vide stražnje strane poligona,
- kod korištenja odbacivanja stražnjih poligona se kroz prazninu vide ostali objekti u sceni,
- te se kod iscrtavanja prozirnih ili poluprozirnih objekata tehnikom praćenja zrake može dogoditi da se ne prepoznaju i ulazak i izlazak zrake iz predmeta, pa se ne provedu svi potrebni izračuni lomljenja zrake.

Neki procesi izrade poligonalnih mreža mogu stvoriti praznine u mrežama, te su stoga potrebni postupci kojima će se mreže s takvim nedostacima popraviti i učiniti prikladnima za svoju namjenu.

II. Nastanak i kategorizacija praznina u mrežama

Attene *et al.* [1, str. 8] navode nekoliko načina na koje u poligonalnim mrežama nastaju praznine koje zahtijevaju popravak: kod skeniranja laserom jedan dio predmeta može zasjeniti njegov drugi dio pa zasjenjeni dio neće biti uzorkovan; kod modela stvorenih CAD alatima se pri pretvorbi matematički zadanih ploha u poligonalne mreže može dogoditi da se rubovi susjednih ploha nakon pretvorbe više ne preklapaju, čime nastaju diskontinuiteti na mjestima gdje je namjera bila modelirati jednu neprekinutu plohu.

Kod imenovanja defekata u poligonalnim mrežama, literatura na engleskom jeziku [1, str. 8] [2, str. 885] razlikuje dvije vrste praznina u mrežama. Razlikuje ih se jer se načini na koje se popravljaju bitno razlikuju.

Razmaci (*gaps*) nastaju kada se rubovi trokutnih mreža ne dotiču iako bi trebali. Primjer razmaka je vidljiv na slici 1b. U takvim su slučajevima praznine omeđene jednim ili više odvojenih lanaca bridova. Neki od uzroka njihovog nastajanja su razlike u parametrima teselacije između ploha, manjak preciznosti korištenih brojevnih tipova, ili ljudska pogreška u modeliranju. Postupak kojim se ovakvi nedostatci uklanjaju naziva se zatvaranje razmaka (*gap closing*), te obično podrazumijeva spajanje postojećih lanaca bridova u jedan, bez uvođenja novih poligona u mrežu. U ovom radu postupci za zatvaranje razmaka neće biti razmatrani.

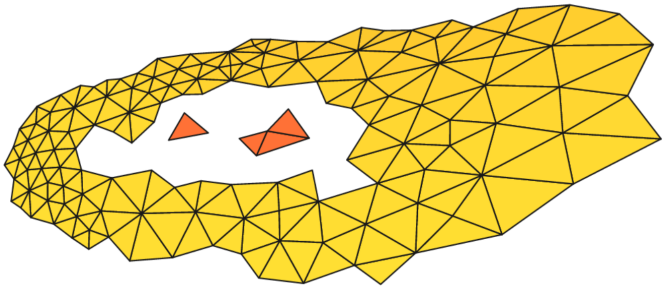
Za razliku od razmaka, rupe (*holes*) su omeđene jednim ili više ciklusa bridova. Rupe obično predstavljaju nedostatak neke veće površine mreže. Unutar rupa se mogu nalaziti i manji dijelovi mreže koji nisu spojeni s ostatkom mreže, te se nazivaju otocima (*islands*). Ukoliko u rupi postoje otoci, takvu rupu omeđuje više od jednog ciklusa bridova, u suprotnom je rupa omeđena samo jednim ciklusom. Na slici 1a je prikazan primjer rupe s otocima. Postupak popravka ovakvih nedostataka se naziva krpanjem rupa (*hole filling*). Rupe je preporučljivo krpati dodavanjem novih poligona u mrežu.

Attene *et al.* [1, str. 18] dodatno razlikuju algoritme kojima je cilj samo zakrpati rupu od onih koji nastoje da krpajuća geometrija

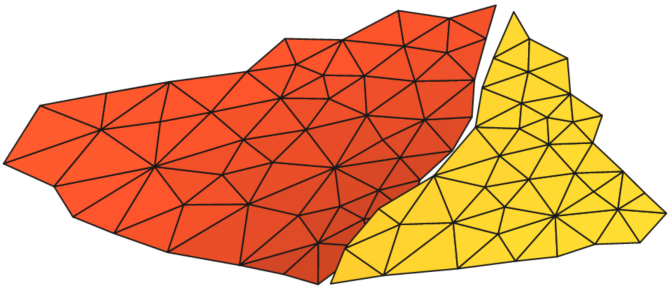
III. Problematika krpanja rupa

Krpanje rupa u poligonalnim mrežama je netrivijalan problem i za njegovo su rješavanje razvijeni brojni pristupi. Pri odabiru algoritma za krpanje rupa je važno uzeti u obzir da algoritmi mogu u mrežu uvesti nove defekte. Na primjer, mreža nakon primjene algoritma više neće imati rupa, ali će možda sijeći sama sebe ili sadržavati trokute koji degeneriraju u dužine. Dio pristupa postojećoj mreži samo dodaje poligone, dok dio mijenjaja i postojeće točke, što može biti nepoželjno. Neki algoritmi mogu broj trokuta u modelu znatno povećati, čak i za dva reda veličine [3, str. 99].

I sam postupak prepoznavanja rupa u mrežama može predstavljati problem za sebe, pogotovo kad je potrebno prepoznati eventualne otoke u rupi. Algoritmi za krpanje rupa koji ne uzimaju otoke u obzir će novim trokutima zatvoriti samo vanjski ciklus bridova, a rubovi otoka će se smatrati odvojenim rupama i bit će zatvoreni u zasebne volumene, odvojene od glavnoga. Kao što je vidljivo na slici 2, kod krpanja rupa rješenje nije jednoznačno određeno.

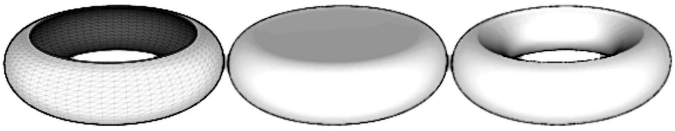


(a) Rupa s otocima



(b) Razmak

Slika 1: Primjeri praznina u mrežama [1, str. 6]

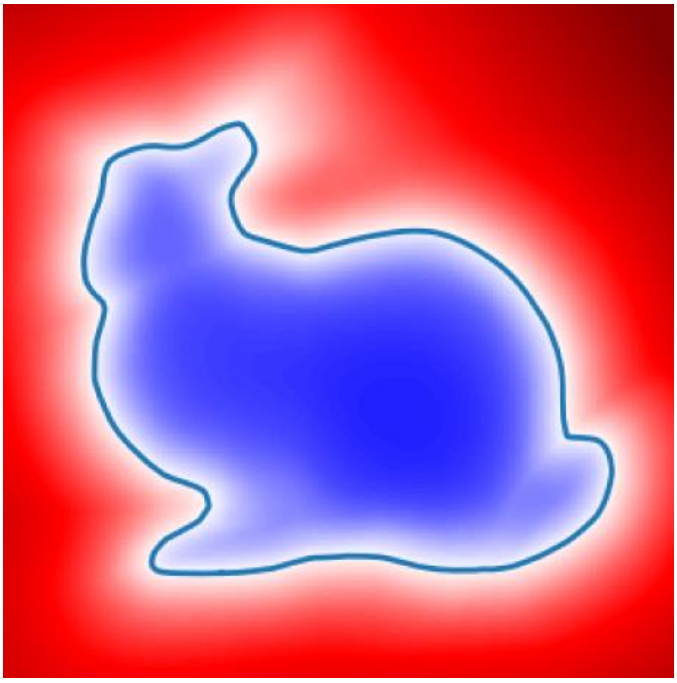


Slika 2: Lijevo je prikazan dio torusa koji je potrebno zakrpati. U sredini i desno su prikazana dva različita rezultata krpanja. Ista metoda može dati oba prikazana rezultata u ovisnosti o parametrima koji joj se zadaju. [2, str. 891]

IV. Usporedbe metoda

Pérez *et al.* u svojem radu [2, str. 897 – 898] uspoređuju 34 metode za krpanje rupa na temelju 19 kriterija. Jedan od kriterija je tip metode: uspoređivane metode su podijelili na pet tipova. [2]

1) Metode temeljene na poligonalnom prikazu rupu grubo zakrpaju poligonima koje zatim prilagode kako bi bolje odgovarali ostatku mreže. Jedan od primjera je metoda koju je 2003. predstavio Liepa [4].
2) Metode temeljene na parametarskom prikazu postojeću geometriju interpoliraju tako da pokušaju pronaći parametarski zadanu plohu koja modelira geometriju koja nedostaje. Primjer ovakve metode je ona koju su 2006. predstavili Branch *et al.* [5].
3) Metode temeljene na funkcijama udaljenosti s predznakom (SDF, *signed distance function, signed distance field*) i volumentrijskim prikazima. SDF modelira udaljenost između točke u prostoru i površine objekta. Točke izvan volumena objekta imaju pozitivnu udaljenost, a točke unutar volumena objekta negativnu udaljenost. Slika 3 prikazuje presjek grafa 3D SDF-a. Jedna od metoda koja se temelji na SDF-ovima je ona koju su 2002. predstavili Davis *et al.* [6]. Neke od volumetrijskih metoda prostor podijele na poliedarske ćelije, na temelju postojeće mreže pokušaju aproksimirati koje od ćelija trebaju biti popunjene, te iz toga iznova grade cijelu mrežu.
4) Metode temeljena na 2D slikama u nekom koraku krpanja koriste dvodimenzionalne projekcije mreža i tehnike za popravak 2D slika koriste za popravak mreža. Ovakvu su metodu 2013. predstavili Lui *et al.* [7].
5) Metode temeljene na kontekstu analiziraju postojeću mrežu i na temelju njezinih svojstava generiraju geometriju za krpanje rupe. Na primjer, metoda koju su 2014. predstavili Vichitvejpaisal i Kanongchaiyos [8] iz modela izdvaja komponente visoke i niske prostorne frekvencije. Mreža niske frekvencije se koristi za grubo krpanje rupe, a zatim se uzorci iz mreže visoke frekvencije prenose na zakrpano područje kako bi ono bolje odgovaralo ostataku mreže.



Slika 3: Graf 2D presjeka 3D SDF-a $d\colon \mathbb{R}^3 \to \mathbb{R}$ za model zeca. Crvena boja označava pozitivnu vrijednost funkcije, a plava negativnu. Svjetlije boje označavaju manju apsolutnu vrijednost. Plava krivulja je rub objekta, na kojem je vrijednost funkcije jednaka nuli. [9, str. 166]

Metode se razlikuju i po uvjetima koji su im potrebni za rad. Neke metode mogu zakrpati i velike rupe, dok ostale daju dobre rezultate samo na rupama kod kojih nedostaje samo manji dio mreže. Jednostavnije metode obično zahtijevaju da se obod rupe približno nalazi u jednoj ravnini, dok složenije tipično podržavaju i krpanje rupa složenijih oblika u prostoru. Neke od metoda ovise o tome da bridovi koji sačinjavaju obod rupe budu homogenih duljina. Neke su prikladnije za glatke površine, dok neke dobre rezultate daju samo kod ploha s

grubom teksturom. [2, str. 893] Mnoge od metoda će uvijek dati rješenje pod cijenom da u mrežu mogu dodati sjecišta mreže same sa sobom. Pristupi koji izbjegavaju stvaranje sjecišta imaju manu da za neke ulaze neće dati rješenje. [1, str. 17]

V. Zaključak

U literaturi je opisan velik broj različitih pristupa za krpanje rupa u poligonalnim mrežama. One se uvelike razlikuju u svojim pristupima, situacijama u kojima daju povoljne rezultate, te u nepravilnostima koje mogu dodati u mrežu. Kod odabira metode je potrebno obratiti pozornost na njenu vremensku složenost, daje li dobre rezultate za konkretnu primjenu, te eventualno i za težinu implementacije ako se ne koristi gotovo rješenje. Postoje radovi koji metode za krpanje rupa uspoređuju u tabličnom obliku. Takvi radovi kao što su [1]–[3] mogu uvelike olakšati odabir metode.

Literatura

[1] M. Attene, M. Campen, and L. Kobbelt, "Polygon mesh repairing: An application perspective," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 1–33, 2013.

[2] E. Pérez, S. Salamanca, P. Merchán, and A. Adán, "A comparison of hole-filling methods in 3d," *International Journal of Applied Mathematics and Computer Science*, vol. 26, no. 4, pp. 885–903, 2016.

[3] X. Guo, J. Xiao, and Y. Wang, "A survey on algorithms of hole filling in 3d surface reconstruction," *The Visual Computer*, vol. 34, no. 1, pp. 93–103, 2018.

[4] P. Liepa, "Filling holes in meshes," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2003, pp. 200–205.

[5] J. Branch, F. Prieto, and P. Boulanger, "A hole-filling algorithm for triangular meshes using local radial basis function," in *Proceedings of the 15th International Meshing Roundtable*. Springer, 2006, pp. 411–431.

[6] J. Davis, S. R. Marschner, M. Garr, and M. Levoy, "Filling holes in complex surfaces using volumetric diffusion," in *Proceedings. First international symposium on 3d data processing visualization and transmission*. IEEE, 2002, pp. 428–441.

[7] L. M. Lui, C. Wen, and X. Gu, "A conformal approach for surface inpainting," *Inverse Problems & Imaging*, vol. 7, no. 3, p. 863, 2013.

[8] P. Vichitvejpaisal and P. Kanongchaiyos, "Surface completion using laplacian transform," *Engineering Journal*, vol. 18, no. 1, pp. 129–144, 2014.

[9] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 165–174.