SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 176

PRIKAZ SIMULACIJE ŠIRENJA DIMA

Sara Sičić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 176

PRIKAZ SIMULACIJE ŠIRENJA DIMA

Sara Sičić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br. 176

Pristupnica:	**Sara Sičić (0036515679)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Prikaz simulacije širenja dima**

Opis zadatka:

Proučiti fizikalnu podlogu koja opisuje pojave pri formiranju dima. Razraditi model koji omogućuje simuliranje širenje dima u zadanom prostoru. Ostvariti implementaciju simulacije te pripadnu vizualizaciju. Ostvariti prikaze rezultata na modelu širenja dima. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti programski jezik programski jezik C# i grafičku programski pogon Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 11. lipnja 2021.

SADRŽAJ

1. Uvod

Svakodnevno smo u doticaju s filmovima, video igricama i raznim drugim digitalnim sadržajima. Takvi sadržaji puni su vizualizacija za koje je zaslužna računalna grafika. Računalna grafika nastoji fizikalno dosljedno prikazati stvarnu okolinu te zadaci računalne grafike često zahtijevaju složenija algoritamska rješenja kako bi postigli uvjerljive rezultate. Jedan od takvih izazova za računalnu grafiku predstavlja i prikaz dima (Slika 1.1). Na slici 1.2 prikazan je primjer simulacije dima u videoigri, a na slici 1.3 vidimo kako se simulacije dima koriste i u filmskoj industriji.

Slika 1.1: Prikaz simulacije dima

Prikaz realistične simulacije dima još uvijek predstavlja velik izazov računalnoj grafici. Glavni razlog tome je što je velik udio ponašanja dima određen kompleksnim međudjelovanjem milijardi sitnih čestica u zraku. Kroz povijest računalne grafike bilo je puno pokušaja simulacije dima, a isprva su bili temeljeni na česticama ili na stazama. No, takvi načini simulacije nisu sposobni u potpunosti prikazati ponašanje fluida izloženog smetnjama. Kako bi se nadoknadili nedostaci takvih metoda, potrebno je simulirati dinamiku fluida.
Cilj ovog završnog rada je istražiti fizikalnu pozadinu širenja dima u stvarnosti, napraviti pregled najpoznatijih metoda simulacija širenja dima i naposljetku implementirati jednu od tih metoda.

Slika 1.2: Prikaz simulacije dima iz cigarete u igrici Cyberpunk 2077

Slika 1.3: Prikaz simulacije dima u filmu Cursed (2005)

2. Fizikalna podloga

2.1. Opis dima

Dim je nakupina malih čestica čvrste ili tekuće tvari rasutih u zraku koje su nastale nepotpunim izgaranjem tvari. U makroskopskim razmjerima dim se ponaša poput poluprozirne tekućine. Stoga ću u svojoj implementaciji simulacije dima pretpostaviti da se plinovi mogu modelirati kao poluprozirni, nestlačivi fluidi konstantne gustoće.

2.2. Navier-Stokesove jednadžbe

Simulacija fluida sastoji se od procjene Navier-Stokesovih jednadžbi koje precizno opisuju kretanje dima tijekom vremena. Ove jednadžbe su nazvane prema Clause-Louise Navieru i George Gabriel Stokesu. Oni su proširili Eulerove jednadžbe tako da je njima moguće opisati viskozne fluide.

Navier-Stokesova jednadžba:

$$\frac{\delta \overrightarrow{u}}{\delta t} = -\overrightarrow{u} \cdot \bigtriangledown \overrightarrow{u} - \frac{1}{\rho} \bigtriangledown \rho + \vartheta \bigtriangledown^2 \overrightarrow{u} + \overrightarrow{F} \tag{2.1}$$

Ova jednadžba izvodi se iz drugog Newtonovog zakona i predstavlja zakon o očuvanju količine gibanja fluida. Komponente od kojih se sastoji ova jednadžba su advekacija ($-\overrightarrow{u} \cdot \bigtriangledown \overrightarrow{u}$), tlak ($-\frac{1}{\rho} \bigtriangledown \rho$), viskoznost ($\vartheta \bigtriangledown^2 \overrightarrow{u}$) i vanjske sile ($\overrightarrow{F}$).

Jednadžba kontinuiteta:

$$\bigtriangledown \cdot \overrightarrow{u} = 0 \tag{2.2}$$

U ovoj jednadžbi $\overrightarrow{u}$ označava brzinu fluida. Ova jednadžba koristi se pri opisu nestlačivih fluida. Pritisak ograničava protok tako da je volumen elemenata fluida konstantan.

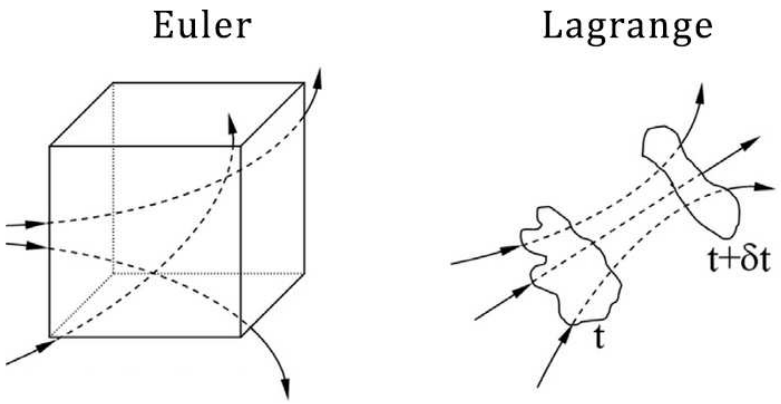
Rješenje ovih jednadžbi je brzina protoka koja je vektorsko polje. Za svaku točku u fluidu, u bilo kojem trenutku, ova jednadžba daje vektor čiji su smjer i veličina brzine fluida u tom prostoru i tom trenutku. Navier-Stokesove jednažbe su u većini slučajeva nelinearne diferencijalne jednadžbe, te ih je u vrlo malom broju slučajeva moguće riješiti egzaktno. Stoga se za rješavanje ovih jednadžbi većinom koriste aproksimacije rješenja primjenom iterativnog postupka. U računalnoj grafici dva najčešće korištena pristupa za rješavanje Navier-Stokesovih jednadžbi su Eulerova metoda i Lagrangeove metoda.

3. Modeli fluida

3.1. Eulerove i Lagrangeove metode

Lagrangeove metode računaju kretanje svake čestice fluida zasebno, čime predstavljaju cijeli fluid kao sustav čestica. Glavni nedostaci ovih metoda su što skaliraju simptotsku složenost O (n logn) te su često manje učinkovite jer se približavaju nestlačivoj granici. Stoga se Lagrangeove metode rijetko koriste za simulaciju dima.

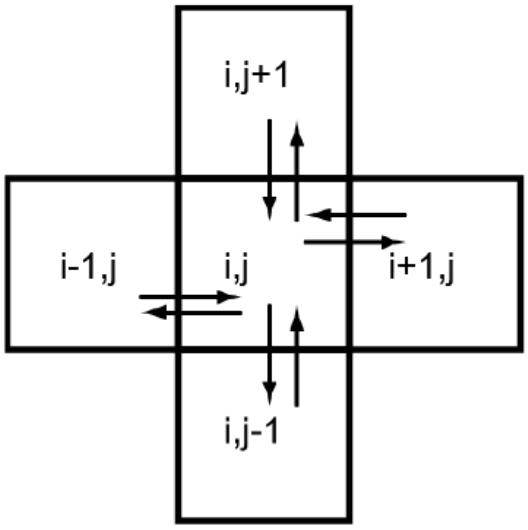
S druge strane, u Eulerovim metodama čestice su dio veće funkcije koja ih sve pokreće. U ovim metodama koriste se mreže fiksnih točaka raspoređenih na području simulacije. Takve rešetke su fiksirane u prostoru, te se tijekom vremena mjenjaju svojstva i količina fluida u ćelijama rešetke. Nedostaci ovih metoda su da izvođenje može biti sporo u slučaju kada je mreža puno veća od promatranog tijela, što čini zahtjevnim postavljanje mreže za složenija tijela te ograničenost fluida na rešetku.
Slika 3.1 ilustrira razliku između ove dvije metode.



Slika 3.1: Usporedni prikaz Eulerove i Lagrangeove metode

3.2. Metoda mreže

Metoda koju sam izabrala implementirati djelomično je temeljena na algoritmu rada Josa Stama. Odabrala sam implementirati ovu metodu jer prioritizira brzinu izvođenja i vjernost prikaza nad fizikalnom točnošću.
Kako bi se postigao konačan prikaz dima, prvo se područje prikaza podijeli na konačan broj identičnih ćelija. U ovom radu opisan je postupak za dvodimenzionalnu simulaciju dima, no isti postupak može se koristiti i za prikaz dima u tri dimenzije. Metoda se temelji na ideji da svaka ćelija sadrži određenu količinu dima, odnosno određene je gustoće koja se tijekom vremena širi među susjednim ćelijama. Pretpostavljamo da ćelija može dijeliti svoju gustoću samo sa četiri susjedne ćelije. (Slika 3.2)



Slika 3.2: Pomoću difuzije ćelija dijeli svoju količinu dima sa susjednim ćelijama

Dvije komponente koje utječu na širenje gustoće dima između ćelija su dolazna i odlazna gustoća. Dolaznu gustoću ćelije predstavlja prosjek odlaznih gustoća njoj susjednih ćelija, a odlazna komponenta difuzije dima opisuje se kao umnožak difuznog koeficijenta i gustoće dima promatrane ćelije. Difuzni koeficijent opisuje brzinu kojom se količina dima ćelije širi prema susjednim ćelijama. Formula koja objedinjuje ove dvije jednadžbe prikazana je pod oznakom (3.1).

$$F_{\mathrm{i,j}} = d \cdot \left(\frac{F_{\mathrm{i,j+1}} + F_{\mathrm{i\text{-}1,j}} + F_{\mathrm{i,j\text{-}1}} + F_{\mathrm{i+1,j}}}{4} + F^{\mathrm{old}} i, j \right) \tag{3.1}$$

Ovim formulama dobit ćemo dim koji se jednako raspršuje u svim smjerovima, no kako bi prikaz bio vjerniji stvarnom kretanju dima, prijenos dima prema gornjoj ćeliji

treba biti veći. To ćemo ostvariti tako da povećamo koeficijent prijenosa gornje ćelije na višu vrijednost od koeficijenata prijenosa ostalih ćelija. Ovime ćemo osigurati da se dim kreće prema vrhu zaslona.

4. Implementacija

Za implementaciju simulacije dima koristila sam grafički programski pogon Unity. Unutar Unity projekta stvorila sam Quad objekt kojeg sam prekrila sa Unlit Shaderom unutar kojeg je implementirana metoda mreže opisana u odlomku 3.2. Unutar sjenčara (engl. *Shader*) količina dima kodirana je u alfa kanal glavne teksture. Definirani su i središte širenja dima te polumjer dima. Funkcija *vert* prima informaciju o položaju vrhova (engl. *vertex*) u koordinatama svijeta te ih pretvara u koordinate zaslona.

Unutar funkcije *frag* u Unlit Shaderu računa se kako pobojati svaki piksel na ekranu, ili u slučaju ove implementacije, s kojom količinom bijele boje. Kod opisan u nastavku napisan je unutar *frag* funkcije u sjenčaru. Sjenčari u Unityu ne podržavaju prikaz mreža, no s obzirom da koristim Quad objekt i da je poznata veličina korištene teksture, program cijelo vrijeme zna izračunati koje piksele trenutačno iscrtava. Pikselima Unity Quada pristupam pomoću quad UV-a te potom prikazujem teksturu pomoću mreže tako da izračunam vrijednost količine dima u središnjoj ćeliji pomoću vrijednosti alpha kanala njenih četiri susjednih ćelija. Zadnja linija koda na slici 4.1. prikazuje kako ćelija gubi gustoću zbog difuzije prema susjednim ćelijama, no također joj se gustoća i povećava zbog gustoća koje joj dolaze od susjednih ćelija. Koeficijent *_Diffusion* predstavlja brzinu širenja dima. Ako uz vrijednosti gustoće svih susjednih ćelija postavimo isti koeficijent, dim će se jednako širiti u svim smjerovima, no s obzirom da znamo da se u stvarnosti dim više širi prema gore, koristimo veći koeficijent *_Transmission* uz gornju ćeliju. Koeficijent *_Diffusion* izvorno je postavljen na 0.09, a koeficijent *_Transmission* na 2, ali oba ova parametra možemo mijenjati unutar Unitya kako bi varirali izgled dima. (Slika 4.1)

```
fixed2 uv = round(i.uv * 128) / 128;
half c = 1 / 128;

float center = tex2D(_MainTex, uv + fixed2(0, 0)).a;
float left = tex2D(_MainTex, uv + fixed2(-c, 0)).a;
float right = tex2D(_MainTex, uv + fixed2(+c, 0)).a;
float top = tex2D(_MainTex, uv + fixed2(-0, -c)).a;
float bottom = tex2D(_MainTex, uv + fixed2(0, +c)).a;

float factor =  _Diffusion * (( top * _Transmission + bottom * 1 + left * 1 + right * 1 ) - (_Transmission + 3) * center);
```

Slika 4.1

Parametar *_Minimum* (izvorne vrijednosti 0.005) predstavlja minimalnu količinu protoka dima. Bez ovog dijela koda male vrijednosti bi se pri izračunu pale na nulu i dim bi se prestao širiti. (Slika 4.2)

```
if (factor >= -_Minimum && factor < 0.0)
    factor = -_Minimum;
center += factor;
```

Slika 4.2

Kako bi se dim uopće mogao širiti, potrebno je postaviti izvor dima. U ovoj implementaciji izvor dima je krug polumjera *_SmokeRadius* (izvorno postavljen na 0.045) čiji je izvor vrh korisnikovog računalnog miša. Količina dima koja se stvara ispod pokazivača postepeno se povećava što se dulje računalni miš drži pritisnutim. Na kraju izračuna količine dima u ćeliji, funkcija *frag* vraća vrijednost boje (koja je u ovoj implementaciji uvijek bijela) i izračunate transparentnosti promatrane ćelije. (Slika 4.3)

```
if (distance(i.wPos, _SmokeCenter) < _SmokeRadius) center += 0.05;

return float4(1, 1, 1, center);
```

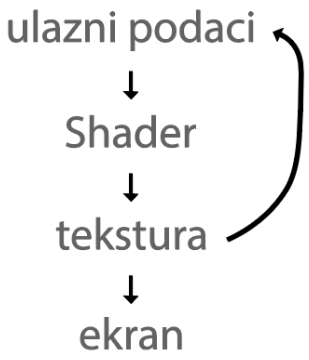
Slika 4.3

Ovaj sjenčar pridružen je Quad objektu, zajedno sa skriptom *MouseSmoke.cs* koja omogućuje da izvor gustoće dima bude korisnikovo pomicanje pritisnutog miša za računalo. (Slika 4.4)

```
void Update()
{
    if (isDragging) {
        Vector3 mousePosition = Input.mousePosition;
        material.SetVector("_SmokeCenter",
            Camera.main.ScreenToWorldPoint(mousePosition)
        );
    }
    else {
        material.SetVector("_SmokeCenter",
            Camera.main.ScreenToWorldPoint(new Vector4(0,0,0,0))
        );
    }
}
```

Slika 4.4: Metoda Update skripte *MouseSmoke.cs*

S obzirom da se inače pri radu sa sjenčarima svaka operacija koju obavljamo unutar sjenčara šalje na zaslon i potom zauvijek gubi, u projekt je dodana dodatna skripta koja to unaprijeđuje za potrebe ove simulacije. Redoslijed izvršavanja operacija unutar programa prikazan je na slici 4.5.

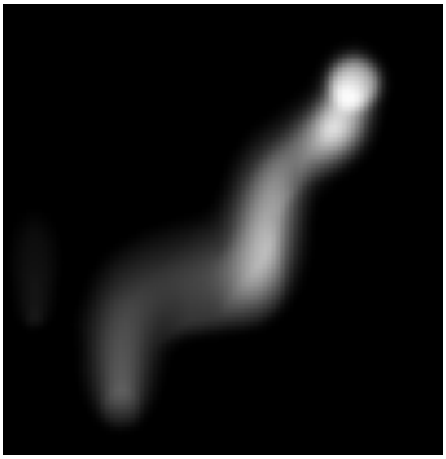


Slika 4.5: Redoslijed izvršavanja operacija

Ovakav postupak će nam omogućiti da izgled mreže prvo prikazujemo na nekoj teksturi i da potom tu teksturu prikažemo na ekranu. Ovo omogućuje sjenčaru da postepeno izgrađuje prikaz, umjesto da se prikaz svaki put izbriše.

5. Rezultati

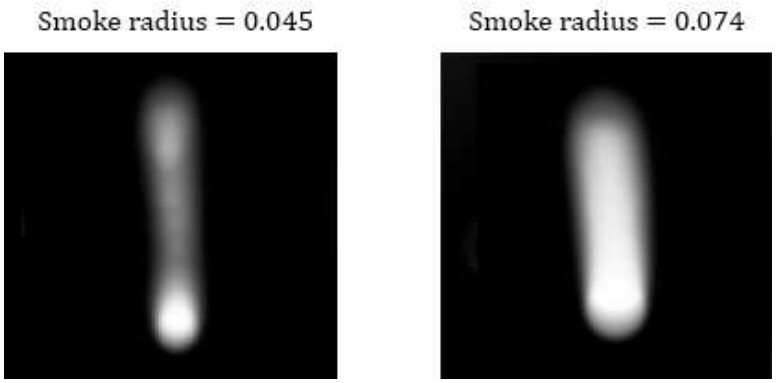
Implementacija je ostvarena u programskom pogonu Unity 2020.3.7f1. Programska implementacija je testenirana na računalu s AMD Radeon RX Vega M GL grafičkom karticom. Rezultat rada je program za simulaciju dima. Držeći miš korisnik ostavlja trag dima po crnom ekranu. (Slika 5.1)



Slika 5.1: Simulacija dima

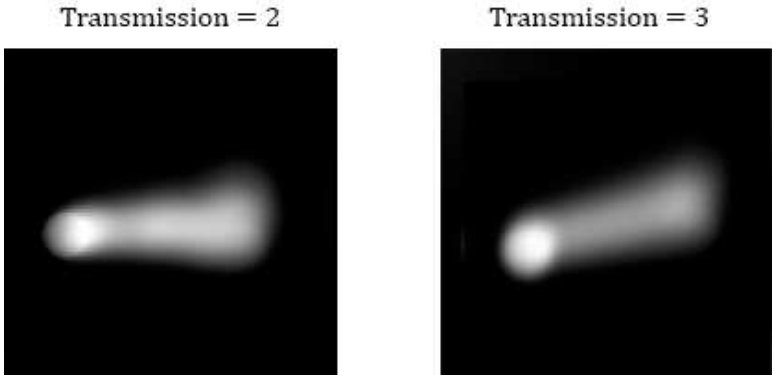
5.1. Utjecaj parametara

Utjecaj četiri parametara koje smo postavili unutar sjenčara znatno utječe na izgled dima unutar simulacije. Na slici 5.2. vidimo kako mijenjajući polumjer dima možemo dobiti širi ili uži trag dima.



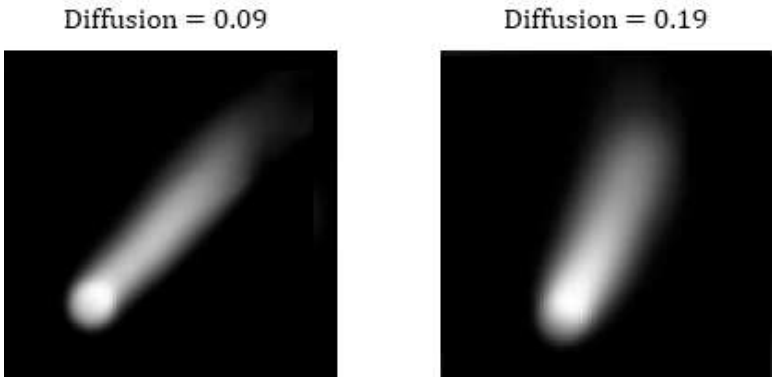
Slika 5.2: Usporedba izgleda dima ovisno o postavljenom polumjeru

Ovisno na koji broj postavimo parametar Transmission, dim će se više ili manje širiti prema vrhu zaslona. Na slici 5.3. vidi se kako će dok se računalnim mišem povuče ravna linija po zaslonu dim više putovati prema vrhu zaslona pri većoj vrijednosti parametra Transmission.



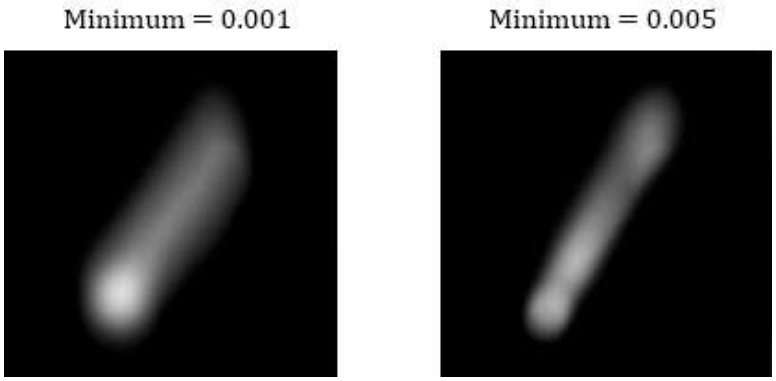
Slika 5.3: Usporedba izgleda dima ovisno o postavljenom parametru Transmission

Parametar Diffusion utječe na brzinu kojom će se dim širiti po zaslonu. Utjecaj ovog parametra vidi se na slici 5.4.



Slika 5.4: Usporedba izgleda dima ovisno o postavljenom parametru Diffusion

Na što nižu vrijednost postavimo parametar Minimum, to će se dulje dim zadržavati na zaslonu. Na slici 5.5 može se primjetiti kako je pri vrijednosti parametra 0.001 dim zamućeniji i prisutniji nego pri višoj vrijednosti, ovo je zato što se dim stigao više raspršiti prije nego što je nestao sa zaslona.



Slika 5.5: Usporedba izgleda dima ovisno o postavljenom parametru Minimum

5.2. Moguća unaprijeđenja

Veliki nedostatak simulacije je niska rezolucija koja je rezultat toga da se kroz simulaciju cijelo vrijeme računa vrijednost svake ćelije na ekranu tako da bi se to isplatilo optimizirati kako bi prikaz bio kvalitetniji. Implementaciju bi također unaprijedilo to da se implementira ovisnost kretanja dima o brzini kretanja računalnog miša kako bi prikaz izgledao realističnije.

6. Zaključak

Računalna grafika se tijekom svog razvoja susretala s mnogim tehnikama simulacije dima. Ovaj rad pruža kratak pregled mogućih metoda i pozadinu simulacije dima i ostalih fluida. Algoritam opisan u ovom radu temeljen je na metodi mreže i implementira osnovne funkcionalnosti difuzije dima, no pruža prostora za detaljniju nadogradnju, ovisno o potrebama digitalnih sadržaja u sklopu kojih bi se koristio.

LITERATURA

1. Jos Stam, *Real Time Fluid Dynamics for Games*, 2003

2. David Cline, David Cardon, Parris K. Egbert,*Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics* 2013

3. Georgijs Sidorovs, *Physically Based Smoke Simulation and Behavior for Mixed Reality*, 2018

Sažetak

Ovaj završni rad opisuje fizikalnu osnovu širenja dima. Pružen je kratak pregled osnovnih metoda simulacije dima i detaljnije je opisan algoritam koji je korišten u implementaciji simulacije dima. Opisan je postupak implementacije u Unityju i prikazani su rezultati izvođenja implementacije simulacije dima te su opisana moguća unaprijeđenja simulacije.

Ključne riječi: simulacija dima, dim, Unity, jednadžbe Navier-Stokes

Display of smoke spread simulation

Abstract

This final paper describes the physical basis of smoke propagation. A brief overview of the basic methods of smoke simulation is provided and the algorithm used in the implementation of smoke simulation is described in more detail. The implementation procedure using Unity is described and the results of the implementation are presented, as well as the possibilities of improving the simulation.

Keywords: smoke simulation, smoke, Unity, Navier-Stokes equations