SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2099

ANIMACIJA FLUIDA U PROŠIRENOJ STVARNOSTI

Tin Salopek

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2099

ANIMACIJA FLUIDA U PROŠIRENOJ STVARNOSTI

Tin Salopek

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 2099

Pristupnik:	**Tin Salopek (0036546319)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Animacija fluida u proširenoj stvarnosti**

Opis zadatka:

Proučiti izradu grafičkih aplikacija na mobilnim uređajima. Posebice obratiti pažnju na izradu animiranih sekvenci. Proučiti mehaniku prolijevanja i prelijevanja tekućina. Razraditi aplikaciju koja simulira prolijevanje tekućine, a ostvarena je u kontekstu proširene stvarnosti, odnosno kao mobilna aplikacija. Načiniti testiranje i usporedbu na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. 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: 23. lipnja 2025.

Sadržaj

1. Uvod

Proširena stvarnost dio je računalne grafike čiji je cilj virtualnim podatcima proširiti prikaz stvarnog svijeta. Zbog toga pripada području Mixed Reality i osim računalne grafike koriste se metode računalnog vida i obrada slike kako bi se iz stvarnog svijeta dobile bitne informacije. Proširena stvarnost ima veliki potencijal za primjenu u područjima razvoja računalnih igara, edukaciji, medicini ili industriji. Osim po području primjene, različite aplikacije koriste različite uređaje kako bi postigle funkcionalnost. Postoje aplikacije koje kao izlazne uređaje za prikaz koriste Head Mounted Display (HMD), dok mobilne aplikacije jednostavno koriste ekran mobilnog uređaja.

Cilj ovog rada je istražiti načine na koje se u aplikacijama proširene stvarnosti mogu animirati fluidi. Prikazane su metode izrade AR aplikacija na mobilnim uređajima. Pokazana su dva načina na koji se može dobiti animacija fluida u sklopu proširene stvarnosti. Prvi način koristi programski sustav Blender i pripadnu podršku za simulaciju fluida i animaciju i tim načinom ne može se dobiti prikaz u stvarnom vremenu. Drugi način može se koristiti za prikaz fluida u nekom spremniku i na taj način dobiva se prikaz u stvarnom vremenu. Međutim to nije fizikalno točan model i iskoristiv je u slučajevima kada je takva aproksimacija fluida dovoljno dobra.

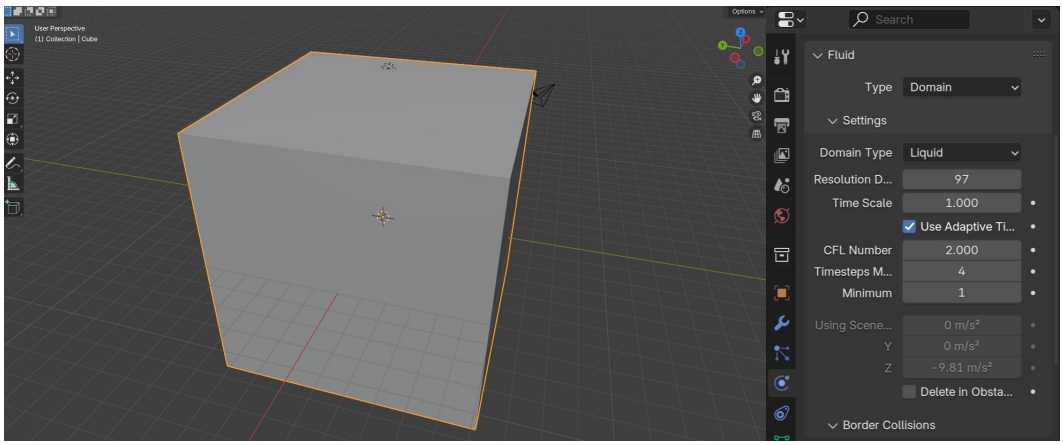
2. Izrada modela objekata i animacije fluida

2.1. Blender

Blender je alat otvorenog koda koji se koristi za 3D modeliranje, animaciju, simulacije, izradu vizualnih učinaka i sličnih stvari. Također je omogućeno kreiranje dodatnih funkcionalnosti kroz Blender Python API.

2.2. Podrška za simulaciju fluida u Blenderu

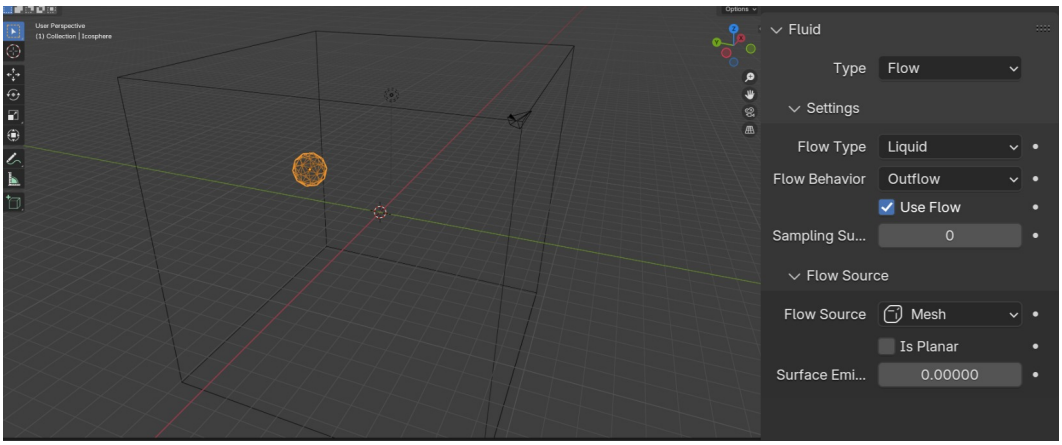
Prvo je potrebno stvoriti i definirati objekte koji će biti tipa *Domain* i *Flow*. To je minimalno potrebno odrediti da bi se pokrenula simulacija u Blenderu. *Domain* je objekt koji definira granice prostora simulacije, a *Flow* je objekt koji emitira ili apsorbira fluid. U praznu scenu dodaje se kocka i dodaje se *Fluid property* s tipom postavljenim na *Domain*. Postavljen je još i *Domain Type* parametar na *Liquid* i povećan *Resolution Divisions* parametar.



Slika 2.1. Postavljanje *Domain* objekta

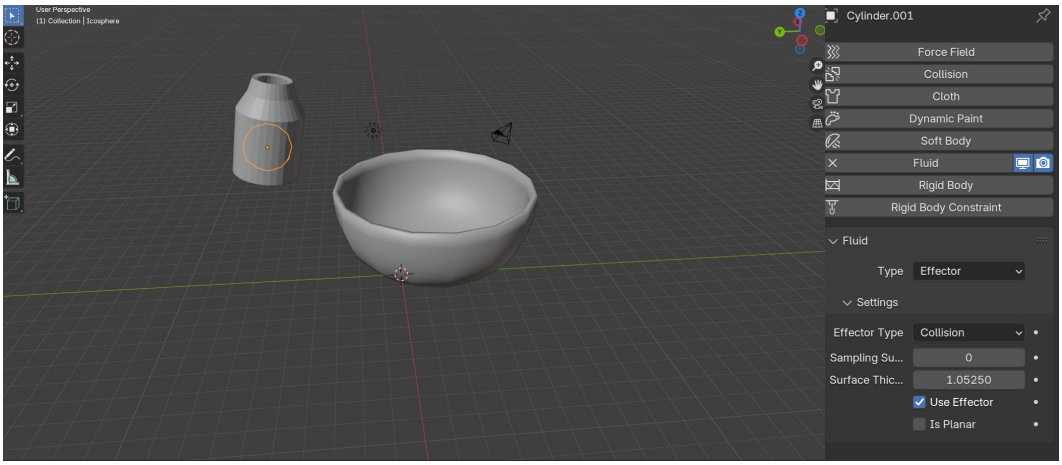
U scenu se dodaje sfera i postavlja se unutar domene. U *Fluid property* se postavlja tip

na *Flow*, *Flow Type* na *Liquid* i *Flow Behaviour* na *Outflow* kako bi taj objekt bio emiter.



Slika 2.2. Postavljanje *Flow* objekta

Prije animacije moguće je modelirati neke objekte s kojima će fluid međudjelovati. Ti objekti dodaju se također unutar domene, a tip im se postavlja na *Effector*. Bitno je i malo povećati parametar debljine objekta u slučaju da fluid prolazi kroz objekte tijekom simulacije.

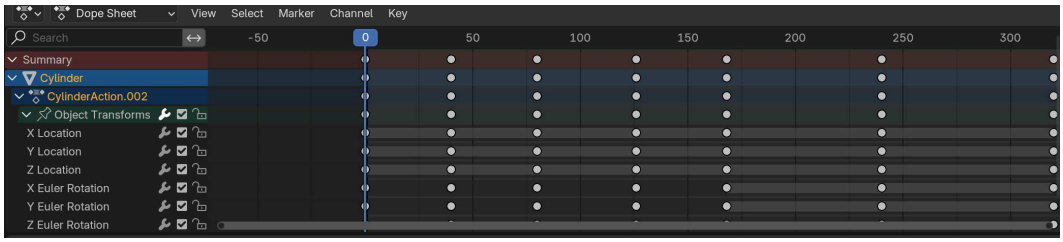


Slika 2.3. Postavljanje *Effector* objekta

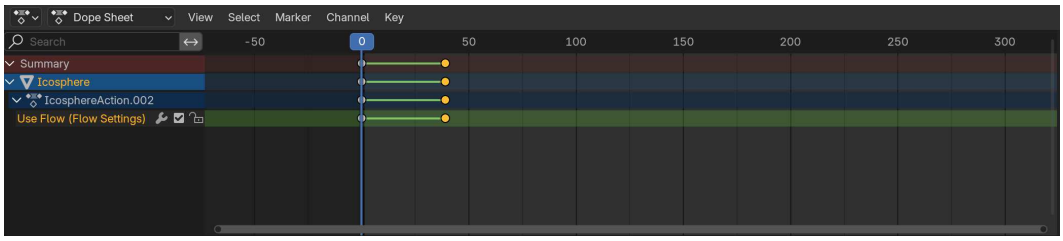
2.3. Animacija

Animacija modela može se napraviti uz Blenderovu podršku za animacije. Koriste se ključni okviri tj. oznake vremena koje sadrže podatke o nekom svojstvu objekta. Ovdje je napravljena jednostavna animacija gdje je mijenjana pozicija i rotacija boce. Također podatak o nekom objektu može biti i nešto osim osnovnih podataka, na primjer pos-

tavljeno je da *Flow* objekt prestane emitirati fluid u nekom ključnom okviru kako bi se realnije prikazao tok fluida iz boce.



Slika 2.4. Postavljanje ključnih okvira na model boce



Slika 2.5. Postavljanje ključnog okvira na Flow objekt

2.4. Izvoz animacije

Za izradu simulacije korišten je Blenderov fizikalni pogon i sada je potrebno na neki način izvesti animaciju u Unity. To je najlakše napraviti tako da se za svaki okvir animacije spreme podatci o mreži poligona u *fbx* datoteku. Moguće je proces automatizirati koristeći Python modul *bpy*. Dakle, prvo se za svaki okvir animacije kopiraju objekti u sceni i dodaju u kolekciju objekata.

```
for i in range(first_frame,last_frame+1):

    my_scene.frame_set(i)

    fluid = all_objects['Object1']
    c_fluid = fluid.copy()
    c_fluid.data=fluid.data.copy()
    bpy.context.collection.objects.link(c_fluid)

    bottle = all_objects['Object2']
    c_bottle = bottle.copy()
```

```
12      c_bottle.data=bottle.data.copy()
13      bpy.context.collection.objects.link(c_bottle)
14
15      bowl = all_objects['Object3']
16      c_bowl = bowl.copy()
17      c_bowl.data=bowl.data.copy()
18      bpy.context.collection.objects.link(c_bowl)
```

Kod 2..1: Kopiranje objekata

Zatim se označe samo novostvoreni objekti kako bi samo njih mogli izvesti. To se isto radi u svakoj iteraciji gornje petlje.

```
bpy.ops.object.select_all(action='DESELECT')

    for obj in [c_fluid, c_bottle, c_bowl]:
        obj.select_set(True)
        bpy.context.view_layer.objects.active = obj
        bpy.ops.object.transform_apply(location=True, rotation=
           True, scale=True)
```

Kod 2..2: Selektiranje objekata

I na kraju za svaki okvir poziva se funkcija *exportscene.fbx()*.

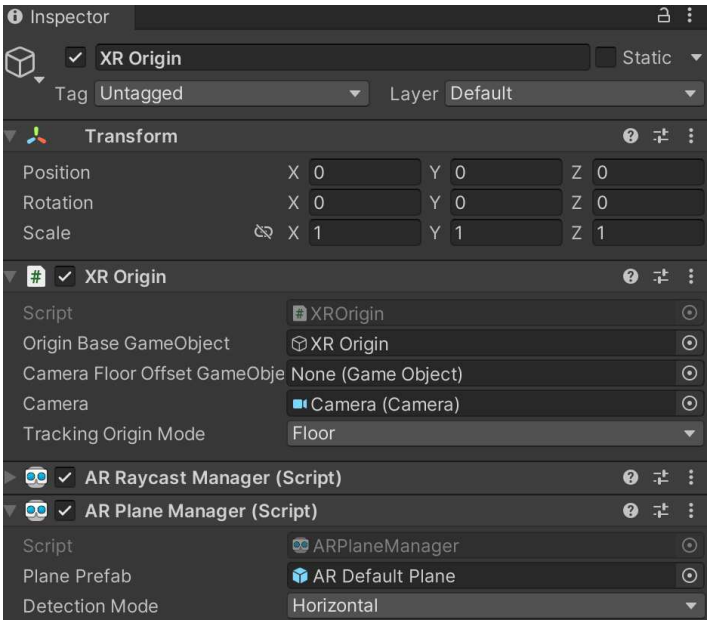
3. Aplikacija proširene stvarnosti

3.1. Programski alat Unity i AR Foundation

U grafičkom pogonu Unity postoji biblioteka AR Foundation i ona omogućava programiranje aplikacija koje koriste proširenu stvarnost na način koji podržavaju više platformi. Te različite platforme su na primjer mobilni operacijski sustavi *Android* i *iOS*, ali i drugi uređaji poput *MagicLeap* i *HoloLens*. Za korištenje AR aplikacije na nekoj platformi potrebno je koristiti Plugin koji to omogućava. Za potrebe ovog rada koristi se ARCore XR Plugin za operacijski sustav Android.

3.2. Interakcija s objektima u sceni

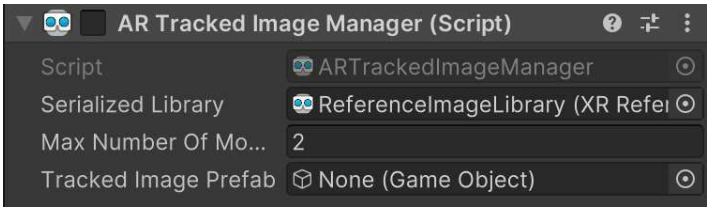
Na početku rada sa scenama uz AR Foundation mora se stvoriti objekt *XR Origin* koji određuje ishodište novog koordinatnog sustava (različitog od globalnog sustava scene) i zove se Tracking sustav. Taj koordinatni sustav u trenutku pokretanja aplikacije potpuno je poravnat s globalnim koordinatnim sustavom scene. Nakon pokretanje, pozicija Tracking sustava može se automatski mijenjati u ovisnosti o detektiranim objektima iz stvarnog svijeta što je nekada korisno. Objekt *XR Origin* mora imati objekt dijete *Camera* koji definira glavnu kameru u sceni. Također, u sceni se nalazi objekt *AR Session* koji upravlja i prati stanje trenutačne AR sjednice. Jedna od bitnijih funckionalnosti koje pruža AR Foundation je korištenje *Managera*. To su *MonoBehaviour* komponente koje implementiraju i omogućavaju nadogradnju nekih AR funkcionalnosti. Na primjer postoji *AR Plane Manager* koji u osnovnoj funkcionalnosti detektira ravne površine u prostoru i vizualizira ih. Nadogradnja funkcionalnosti obično se postiže pretplaćivanjem na događaje koje oni signaliziraju. Na *XR Origin* moguće je dodati neke komponente tipa *Manager* kao što je vidljivo na slici 3.1.



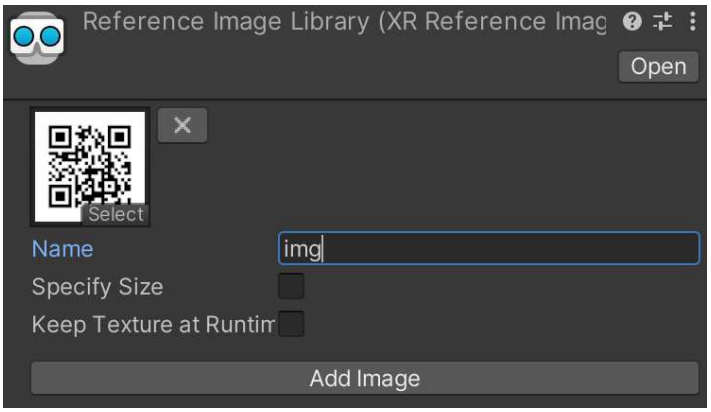
Slika 3.1. Konfiguracija XR Origin objekta

3.2.1. Postavljanje objekata

Postavljanje objekata u AR okruženju moguće je postići na dva načina. Prvi način je korištenjem markera. Markeri su slike koje kamera može prepoznati i na taj način te slike služe kao referentne točke za postavljanje virtualnih objekata. Ovakav način rada zove se *Marker-based AR*. Ova funkcionalnost jednostavno se može postići na način da se na XR Origin objekt doda *AR Tracked Image Manager* (slika 3.2.) i definiraju se slike koje se žele koristiti kao markeri (slika 3.3.). Zatim se pretplati na događaj *trackedImagesChanged* i napravi se logika instanciranja. Ovaj način dobar je ako postoji veća količina objekata za stvaranje na raznim mjestima.



Slika 3.2. AR Tracked Image Manager komponenta



Slika 3.3. Primjer markera

Drugi pristup je nešto fleksibilniji jer ne zahtjeva od korisnika da ima uz sebe prethodno definirani marker i to je *Markerless AR*. Problem postavljanja objekata rješava se na druge načine, na primjer postavljanjem objekata na prepoznate ravnine u prostoru (podove, zidove, ...).

Kako bi se omogućilo postavljanje objekata na detektirane površine potrebno je na XR Origin objekt postaviti *AR Plane Manager* komponenetu i odrediti Prefab koji će se koristiti kako bi se vizualizirale detektirane površine. Na tom Prefabu postavi se Tag na „Plane“ što se koristi u skripti ObjectPlacement.cs (kod 3..1). Ovdje je prikazana metoda *Update* u kojoj se prolazi kroz listu svih Touch struktura i računaju se parametri koji opisuju zraku tj. ray objekt. Klasa *Ray* ima atribute izvor i vektor smjera. U ovom slučaju sve zrake imati će izvor u očištu, a smjer prema određenom pixelu u projekcijskoj ravnini. Nakon što su izračunati parametri zrake provjerava se sječe li zraka neki objekt koji ima *Tag* = *„Plane“*. U slučaju da presjecište postoji instancira se objekt na toj poziciji.

```
1 void Update()
2 {
3     foreach (var touch in Touch.activeTouches)
4     {
5         Vector2 touchPosition = touch.screenPosition;
6         Ray ray = Camera.main.ScreenPointToRay(touchPosition);
7         RaycastHit hit;
8         if (Physics.Raycast(ray, out hit))
9         {
```

```
10              if (hit.collider.CompareTag("Plane") && objPlaced ==
                    false)
11              {
12                  Vector3 hitPoint = hit.point;
13                  float objectHeight = renderer.bounds.size.y;
14                  hitPoint.y += objectHeight / 2f;
15                  Instantiate(obj, hitPoint, obj.transform.
                        rotation);
16
17                  objPlaced = true;
18              }
19          }
20      }
21  }
```

Kod 3..1: Postavljanje objekata

3.2.2. Pomicanje objekata

Kako bi se pomaknuo objekt na novu poziciju prolazi se kroz listu Touch strukture i uzima se prvi dodir koji je korisnik napravio na ekranu. Potom se računa zraka koja ima izvor u očištu a smjer prema koordinatama dodira. Budući da je ideja da se objekt pomiče u x-z ravnini tj. y koordinata će uvijek biti jednaka, računa se za koji parametar t će neka zraka imati y koordinatu jednaku y koordinati objekta. Na taj način može se onda izračunati kolike se nove koordinate x i z. Na kraju je potrebno linearno interpolirati stari položaj objekta i novi kako bi se dobio glađi prijelaz između njih. Razlog je zato što se direktno modificira komponenta transform, i ne koristi se fizikalni sustav ugrađen u Unity te je moguće da objekt prenaglo mijenja poziciju.

```
1  void Update()
2  {
3      foreach (var touch in Touch.activeTouches)
4      {
5          if (touch.finger.index == 0)
```

```
6           {
7               Vector2 touchPosition = touch.screenPosition;
8               Ray ray = Camera.main.ScreenPointToRay(touchPosition
                    ;
9               Vector3 startPosition = ray.origin;
10              Vector3 direction = ray.direction;
11              float t = (transform.position.y - startPosition.y) /
                     direction.y;
12              Vector3 position = new Vector3(startPosition.x + t *
                     direction.x, transform.position.y, startPosition.
                    z + t * direction.z);
13              transform.position = Vector3.Lerp(transform.position
                    , position, Time.deltaTime * movementSpeed);
14          }
15      }
16  }
```

Kod 3..2: Pomicanje objekata

3.3. Uklanjanje skrivenih objekata

Uklanjanje skrivenih objekata najlakše je postići koristeći komponentu *AROcclusionManager* koja se postavlja kao komponentu na glavnu kameru. Uspješnost te metode ovisi o karakteristikama uređaja na kojemu se pokreće AR aplikacija. Uvjet da bi skripta uopće radila je da kamera na uređaju posjeduje kameru koje može registrirati dubinu okoliša. Zbog toga AR Foundation omogućava da se konfigurira ta komponenta u ovisnosti o postojanju takve kamere, željenoj kvaliteti i brzini skrivanja.

Postoji i drugi pristup koji bi se mogao koristiti kod skrivanja objekata, ali uz ograničenje da omogućava skrivanje samo iza prethodno detektiranih ravnina. To se postiže tako da se na Prefab koji služi za vizualizaciju ravnina postavi materijal s sjenčarom koji će prije iscrtavanja ostale geometrije u sceni napuniti Z-spremnik sa svojim z vrijedonstima. U sjenčaru se definiraju naredbe *Zwrite On, ColorMask 0, Ztest LEqual* da bi se ta funkcionalnost omogućila. Mora se dodati i oznaka *Queue = Geometry-1* da se to prikaže prije druge geometrije u sceni. Prednost ovog pristupa je brzina i to što ne mora postojati po-

sebna podrrška za dubinu, a mana je što u praksi sustav za prepoznavanje ravnina ne radi savršeno.

3.4. Osvjetljenje i sjene

U radu s proširenom stvarnosti cilj je osvjetliti virtualne objekte na način koji je u skladu sa svijetlosti iz stvarnog svijeta. Način na koji se to ostvaruje u ovom sustavu je korištenjem *frameReceived* događaja koji se nalazi u klasi *ARCameraManager*. Taj Event poziva pretplaćene funkcije svaki okvir. Pretplatom na taj događaj moguće je pristupiti strukturi ARCameraFrameEventArgs u kojoj se između ostalog nalaze podatci koje kamera prikuplja a govore o svjetlosti iz stvarnog svijeta. Dakle, u glavni izvor svjetlosti definiran u sceni u Unity-u postave se oni parametri svjetlosti koje daje *ARCameraManager* (kod 3..3).

```
void FrameChanged(ARCameraFrameEventArgs args)
{
    ARLightEstimationData lightData = args.lightEstimation;
    if (lightData.mainLightColor.HasValue)
    {
        light.color = lightData.mainLightColor.Value;
    }
    if (lightData.averageMainLightBrightness.HasValue)
    {
        light.intensity = lightData.averageMainLightBrightness.
            Value * 2f;
    }
    if (lightData.mainLightDirection != null)
    {
        light.transform.rotation = Quaternion.LookRotation(
            lightData.mainLightDirection.Value);
    }
}
```

Kod 3..3: Postavljanje parametara osvjetljenja

Zatim se samo koriste materijali koji imaju već ugrađen rad sa svijetlom ili je moguće te izračune napraviti samostalno. Ovo je dovoljno da za promjenu intenziteta i boje objekata, ali i sjene se mogu vidjeti na virtualnim objektima. Moguće je napraviti da objekti stvaraju sjene na detektiranim površinama. Problem koji stoga treba riješiti je taj da detektirane površine trebaju biti transparentne, a u *URP* protočnom sustavu na transparentnim objektima se ne prikazuju sjene. Zato je potrebno ručno u sjenčaru za prikaz detektiranih površina iskoristiti mogućnosti rada s mapom sjene. Prvo se u sjenčaru vrhova poziva funkcija *GetShadowCoord* koja vraća koordinate u mapi sjene na koje se mapira taj vrh, z vrijednost koja predstavlja udaljenost vrha od izvora i homogenu koordinatu. Zatim se sjenčar fragmenata napravi prema kodu 3..4

```
half4 frag (v2f i) : SV_Target
{
    float shadowAmount = MainLightRealtimeShadow(i.shadowCoords)
        ;
    return half4(0, 0, 0, 1-shadowAmount);
}
```

Kod 3..4: Sjenčar fragmenata za primanje sjena

U njemu se poziva funkcija *MainLightRealtimeShadow* koja interno uspoređuje z vrijednost od vektora kojeg je vratila funkcija GetShadowCoords s vrijednosti dubine u mapi sjene. *MainLightRealtimeShadow* vraća "količinu" sjene. Vraća se crna boja, ali u slučaju sjene alfa vrijednost je 1, a u slučaju bez sjene alfa je 0 i sjena se ne vidi, tj. površina je transparentna.

3.5. Pokretanje animacije

Kako bi se animacija pokrenula prilikom postavljanja objekta potrebno je prije Update metode spremiti sve mreže poligona u listu kako se ne bi moralo u svakom okviru pretraživati datoteke u sustavu. To se radi u metodi Start.

```
void Start()
{
    for (int i = startFrame; i <= endFrame; i++)
```

```
    {
        string fileName = $"NewMeshes/anim{i}";
        GameObject fbxObject = Resources.Load<GameObject>(
            fileName);
        List<Mesh> meshes = new List<Mesh>();
        foreach (MeshFilter currMesh in fbxObject.
            GetComponentsInChildren<MeshFilter>())
        {
            meshes.Add(currMesh.sharedMesh);
        }

        meshesPerFrame.Add(meshes);
    }
}
```

Kod 3..5: Metoda Start u skripti za pokretanje animacije

Zatim se u *Update* metodi provjerava je li prošlo vrijeme trajanja jednog okvira i u tom slučaju se *MeshFilter* komponentama objekta promijeni trenutna mreža poligona.

```
void Update()
{
    timer = timer + Time.deltaTime;
    if (timer > 1f / fps)
    {
        bottle.mesh = meshesPerFrame[currentFrame][2];
        bowl.mesh = meshesPerFrame[currentFrame][0];
        fluid.mesh = meshesPerFrame[currentFrame][1];
        currentFrame = (currentFrame + 1) % (endFrame -
            startFrame + 1);
        timer = 0f;
    }
}
```

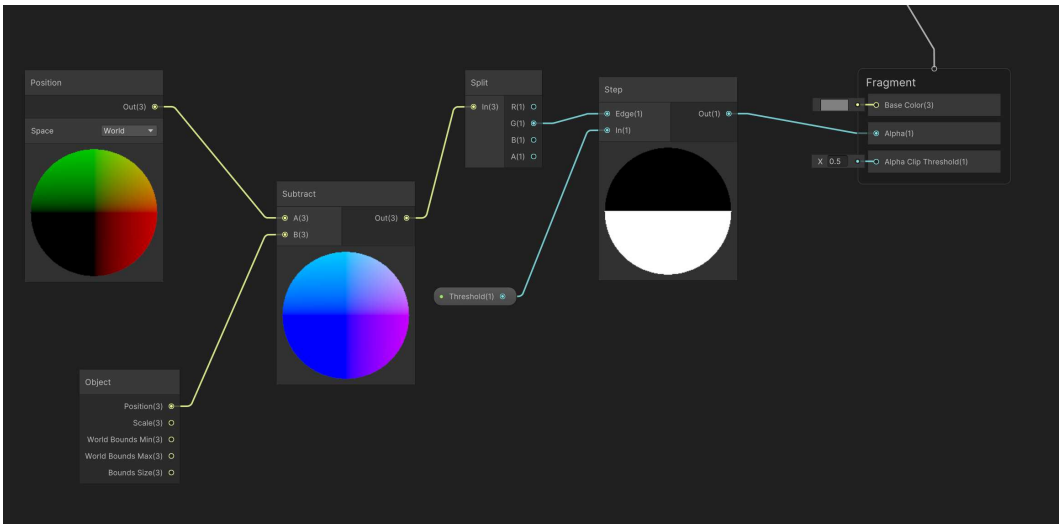
Kod 3..6: Metoda update u skripti za animaciju

4. Interaktivni prikaz fluida

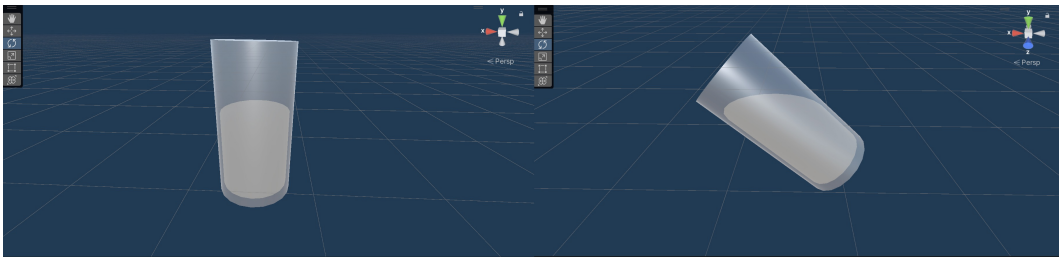
Vrlo često u određenim područjima gdje su potrebne animacije i simulacije fluida potrebno je u stvarnom vremenu prikazati fluide. Najkvalitetnije simulacije su one fizikalno točne no implementacija takvog sustava često je prekompleksna i vremenski prezahtjevna u primjeni. Zbog toga je potrebno na neki drugi način pokušati što vjernije opisati ponašanje fluida. Neke računalne igre primjerice koriste sustave čestica kako bi prikazali fluide koji se prolijevaju, a u slučaju da je potrebno prikazati fluide koji su u boci ili nekom spremniku moguće je bez ikakve dodatne geometrije (uz spremnik) prikazati fluid u spremniku. Takva metoda pokazana je u nastavku.

4.1. Sjenčar fragmenata

Glavna ideja je da se napravi objekt koji predstavlja bocu i unutar tog objekta objekt koji predstavlja fluid i ima isti oblik kao i boca. Na fluid se postavi materijal sa sjenčarom koji u ovisnosti o y koordinati fragmenta fluida postavlja alfa vrijednost na 1 ili na 0. Dakle, od pozicije fragmenta u globalnom koordinatnom sustavu prvo se oduzmu koordinate središta objekta kako bi uvijek promatrali iste y vrijednosti tj. kako se razina fluida ne bi mijenjala s promjenom y vrijednosti objekta. Taj sjenčar najjednostavnije je napraviti pomoću *Shader Graph* alata u Unity-u i prikazan je na slici 4.1. Rezultati primjene takvog materijala su na slici 4.2.

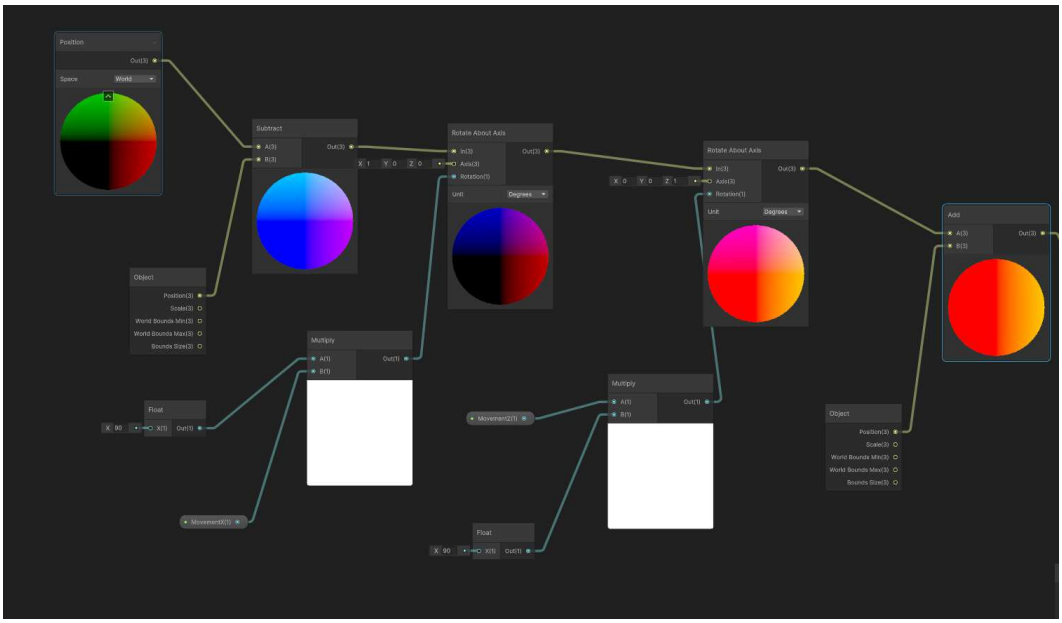


Slika 4.1. Dio sjenčara fragmenata za odsijecanje objekta



Slika 4.2. Prikaz rotacije objekta oko svoje osi

Sada je potrebno na omogućiti da se razina fluida mijenja u ovisnosti o vrijednostima koje će sjenčar dobivati iz skripte koja računa akceleraciju objekta. Prvo se dovede središte lokalnog koordinatnog sustava objekta u središte globalnog sustava te potom rotira objekt fluida oko x osi a zatim i oko z osi. Broj stupnjeva rotacije ovisit će o ulaznim parametrima. Zatim se vrate koordinate u globalni koordinatni sustav zbrajanjem Porta Position čvora Object. I sada te izračunate koordinate mogu se proslijediti umjesto čvora Position na slici 4.1.



Slika 4.3. Dio sjenčara fragmenata za rotaciju objekta

4.2. Nadogradnja skripte za pomicanje modela

U skripti za pomicanje objekta potrebno je ostaviti prijašnju logiku pomicanja i dodati izračun akceleracije i slanje tih vrijednosti do sjenčara.

```
Vector3 velocity = (transform.position - previousPosition) /
    Time.deltaTime;

Vector3 realAcceleration = (velocity - previousVelocity) / Time.
    deltaTime;
smoothedAcceleration = Vector3.Lerp(smoothedAcceleration,
    realAcceleration, Time.deltaTime * smoothSpeed);

acceleration = new Vector3(smoothedAccel.x, 0f, smoothedAccel.y)
    ;
previousVelocity = velocity;
previousPosition = transform.position;
```

Zatim se u trenutku kad je korisnik završio pomicanje mora zapamtiti vrijednost akceleracije i od tog trenutka računati promjena površine fluida po prigušenoj sinusoidi kako

bi se poboljšao privid inercije.

```
float frequency = 6f;
float damping = 1f;

float t = Time.time - stopTime;
acceleration.x = stopValue.x * Mathf.Cos(t * frequency) * Mathf.
    Exp(-t * damping);
acceleration.z = stopValue.z * Mathf.Cos(t * frequency) * Mathf.
    Exp(-t * damping);
```

Parametri koje sjenčar treba mogu se predati dohvaćanjem reference na Renderer komponentu objekta i dohvaćanjem svojstva material. Predaja se radi naredbama:

```
mat.SetFloat("_MovementX", acceleration.x);
mat.SetFloat("_MovementZ", acceleration.z);
```

5. Rezultati

Ovdje su prikazane snimke zaslona tijekom animacija:

Slika 5.1. Objekti na početku animacije

Slika 5.2. Objekti tijekom animacije

Slika 5.3. Fontana

Slika 5.4. Interaktivni prikaz

6. Zaključak

Proširena stvarnost tehnologija je koja je se i dalje razvija i koja ima potencijal biti primjenjena u mnogim područjima. Postoje još mnoge mogućnosti za poboljšanje te tehnologije, od kojih su neke u domeni računalne grafike, a druge u domeni računalnog vida. U domeni računalne grafike glavni cilj je najčešće ostvariti kvalitetan fotorealističan prikaz na uređajima koji nemaju veliku količinu računalne snage.

Cilj ovog rada bio je ostvariti mobilnu aplikaciju proširene stvarnosti koja prikazije animirane fluide. To je postignuto izradom modela i animacija u Blenderu, i zatim njihovo korištenje u Unity-u uz biblioteku AR Foundation. Diskutirane su najbitnije stvari vezane uz izradu AR aplikacija, kao što su interakcija s postavljenim objektima, skrivanje objekata i osvjetljenje. Također, s ciljem ostvarivanja veće razine interakcije s korisnikom implementirana je metoda za prikaz fluida u stvarnom vremenu pomoću sjenčara fragmenata.

Mogućih proširenja aplikacije i rada ima mnogo. Moguće je izraditi više modela u Blenderu i ostvariti realističniji prikaz i animaciju. Poboljšanje bi bilo i kada bi se implementirala fizikalna simulacija fluida. Onda bi se omogućio prikaz u stvarnom vremenu i potencijalno bolji rezultati. Što se tiče sjenčara fragmenata trenutno ne postoje nikakvi površinski efekti na fluidu u boci i to bi bilo nešto što se može poboljšati u tom dijelu. Korištenjem biblioteke OpenXR proces izrade aplikacije bio bi nešto zahtjevniji, ali postojala bi veća mogućnost kontrole nad cijelom izradom prikaza.

7. Literatura

[1] Unity AR Foundation Documentation.

[2] AR Core Documentation.

[3] Blender Documentation.

[4] Youtube video izrade modela fontane.

[5] Liquid shader in Unity

[6] R. Silva, J. C. Oliveira, and G. A. Giraldi, Introduction to Augmented Reality, National Laboratory for Scientific Computation, Petropolis-RJ, Brazil.

Sažetak

Animacija fluida u proširenoj stvarnosti

Tin Salopek

U sklopu rada na aplikacijama proširene stvarnosti može se javiti potreba za prikazom animiranih fluida. U ovom radu pokazan je način kako postići animacije fluida u proširenoj stvarnosti za mobilne uređaje. Modeli koji se koriste i sama animacija napravljena je u programskom alatu Blender. Istražene su i implementirane osnovne funkcionalnosti koje aplikacija proširene stvarnost može imati. To uključuje postavljanje objekata i interakciju s njima, skrivanje skrivenih objekata, osvjetljenje objekata i bacanje sjena. Pokazan je i način na koji se može prikazati fluid u stvarnom vremenu bez fizikalno točnih simulacija.

Ključne riječi: proširena stvarnost; animacija fluida; Unity ; Blender

Abstract

Fluid animation in augmented reality

Tin Salopek

While working on augmented reality applications there may be a need to display animated fluids. In this paper it is shown how to implement animations of fluids in augmented reality for mobile devices. Models and animation were first made using Blender. Basic funcionaities of augmented reality applications were explored and implemented. That includes instantiating and interacting with objects, occlusion, lightning and casting shadows. A method is presented for displaying fluids real-time but without physical accuracy.

Keywords: augmented reality; fluid animation; Unity; Blender

Privitak A:

Link na izvorni kod.