ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**ІМПЛЕМЕНТАЦІЯ МОДЕЛЕЙ ЗАТІНЕННЯ AMBIENT OCCLUSION ДЛЯ ВІЗУАЛІЗАЦІЇ 3D-СЦЕН**»

Виконав: студент 5 курсу, групи ППЗ–51

Спеціальності:

121 Інженерія програмного забезпечення

Пояркін І.С.

Керівник: Марченко В.В.

Рецензент_____

Київ-2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ О.В. НЕГОДЕНКО

«____» _____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

ПОЯРКІН ІГНАТІЙ СЕРГІЙОВИЧ

1.Тема роботи: «Імплементація моделей затінення ambient occlusion для візуалізації 3d-сцен»

Керівник роботи: Марченко В.В.

затверджені наказом вищого навчального закладу від «12» березня 2021 року №65.

2. Строк подання студентом роботи «1» червня 2021 року

3. Вхідні дані до роботи: мова програмування C++, набір Application Programming Interface DirectX, мова програмування HLSL, науково-технічна література та наукові статті з питань, пов’язаних з комп’ютерною графікою.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Дослідження основної проблеми комп’ютерної графіки та методів її вирішення.

4.2. Основні підходи та методи апроксимації в комп’ютерній графіці.

4.3. Дослідження проблем real-time графіки та ambient occlusion.

4.4. Алгоритм screen-space ambient occlusion та методи його удосконалення.

5. Перелік графічного матеріалу:

5.1. Опис основної проблеми рендеру.

5.2. Апроксимація та розв’язок проблеми рендеру для real-time графіки

5.3. Алгоритм Screen-space ambient occlusion

5.4. Удосконалення результату та часу виконання алгоритму SSAO.

5.5. Алгоритм Bilateral Gaussian Blur.

5.6. Фінальне зображення «до» та «після» застосування SSAO

6. Дата видачі завдання: «19» квітня 2021

КАЛЕДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.21	Виконано
2	Розробка архітектури додатку	23.04.21	Виконано
3	Розробка візуалізації 3D-сцени	27.04.21	Виконано
4	Дослідження алгоритмів SSAO, методу Монте-Карло	2.05.21	Виконано
5	Імплементація алгоритму SSAO	15.05.21	Виконано
6	Імплементація алгоритму Bilateral Gaussian Blur та поєднання усіх алгоритмів для виводу фінального зображення	19.05.21	Виконано
7	Розробка обов’язкових демонстраційних матеріалів	21.05.21	Виконано
8	Вступ, висновки, реферат	23.05.21	Виконано
9	Попередній захист роботи	30.05.21	

Студент_____ Пояркін І.С.

Керівник роботи _____ Марченко В.В.

РЕФЕРАТ

Текстова частина бакалаврської роботи: 50ст., 29 рис., 14 джерел.

Ключові слова: ambient occlusion, screen space ambient occlusion (ssao), blur, bilateral gaussian blur, deferred rendering, lambertian reflectance, sample kernel, spectral radiance. альбедо, апроксимація, затемнення сцени, модель затінення blinn-phong, рівняння рендеру, текстура.

Об’єкт дослідження – Ambient Occlusion в комп’ютерній графіці

Предмет дослідження – Screen space ambient occlusion

Мета роботи – дослідження оптимального способу затемнення 3D-сцени за допомогою Screen space ambient occlusion.

Методи дослідження – методи апроксимації розрахунку поведінки світла на 3D-сцені з 3D-об’єктами, метод Монте-Карло для взяття інтеграла по півсфері, метод хаотичного повороту для sample kernel (набору точок всередині півсфери), метод затінення по Блін-Фонгу (Blinn-Phong), метод розділення Gaussian blur на два етапи (горизонтальний та вертикальний) для оптимізації обчислень.

В результаті дослідження: встановлено, що метод затінення по Блін-Фонгу дає змогу правдоподібно та, найголовніше, швидко затінити сцену, враховуючи одне джерело світла; визначено, що з використанням алгоритму deferred rendering SSAO займає менше часу, так як для затінення сцени використовуються ті самі текстури та дані, що потрібні для SSAO; досліджено, що за допомогою різних апроксимацій SSAO можна досягнути правдоподібного відображення сцени в місцях дотику різної геометрії за короткий проміжок часу (для того, щоб встигнути рендерити сцену в 60 кадрів в секунду)

ЗМІСТ

ВСТУП

В сучасному світі комп’ютерна графіка відіграє важливу роль та займає почесне місце серед комп’ютерних наук, що стрімко розвиваються в геометричній прогресії уже впродовж десятки років. Комп’ютерна графіка використовується в багатьох сферах: від медіа (фільми, ігри) до медицини. Завдяки стрімкому розвитку комп’ютерних наук в цілому, потужність процесорів та відеокарт зростає, завдяки чому комп’ютерна графіка становиться все реалістичнішою.

Нажаль, фізично правильно в реальному часі передавати картинку за допомогою комп’ютерної графіки все ще неможливо, так як тільки для освітлення потрібно вираховувати безкінечну кількість променів та їх поведінку в оточенні різних об’єктів.

Саме тому для передачі світу через комп’ютерну графіку використовуються різного роду апроксимації, наближення, що допомагають правдоподібно передати реальність через графіку.

Одною с таких апроксимацій є ambient occlusion. Ambient occlusion використовується для того, щоб надати картинці більш «реальний» вигляд, симулюючи поведінку світла в місцях дотику різних поверхонь.

Ідея ambient occlusion-у з’явилася досить давно, але використовувати дану техніку в realtime rendering (рендерингу в реальному часі) було досить важко, так як вона потребувала занадто багато потужності відеокарти та процесору, що було на той час неможливо. Але для використання ambient occlusion в real-time rendering була придумана техніка, що називається SSAO (Screen space ambient occlusion), або ambient occlusion екранного простору. Вперше цю ідею запропонували та опублікували комп’ютерні науковці з

компанії Crytek у 2007 році. Вони використовували цю техніку для їх гри Crysis, яка била всі рекорди по «реалізму» в той час.

Після цього багато хто із ігрових студій взялися використовувати дану техніку, яка де-факто стала однією із основних для ігрової індустрії.

Основою інформаційної бази диплому стали статті та роботи програмістів компанії Nvidia, які посвятили даній темі багато часу. Вони працювали як над SSAO, так і придумали нові, більш специфічні техніки, такі як HBAO (Horizontal Based Ambient Occlusion), HBAO+, VXAO (voxel ambient occlusion), SSDO (Screen Space directional occlusion). Також статті на цю тему написав Joey de Vries, John Chapman, їх роботи та статті послужили основою даної дипломної роботи.

Об’єктом дослідження є модель затінення Ambient Occlusion, що дозволяє додати «реальності» комп’ютерному зображенню. Предметом дослідження є підвид ambient occlusion – SSAO, або screen space ambient occlusion, як апроксимація поведінки світла в затемнених кутках та місцях, де світло не може потрапити на поверхню зі всіх сторін.

Існує велика кількість різних методів апроксимації для SSAO, та для кожного проекту вони можуть мати свої власні налаштування та реалізації, залежно від того, який результат очікується. Для мультфільмів може використовуватись спеціальний фільтр, що надає картинці своєрідний відтінок, для фільмів ambient occlusion має бути якомога фізично коректніше, для ігор він повинен надавати правдоподібний результат, але розрахунки мають бути дуже швидкими.

Метою даної роботи є написання алгоритму SSAO та візуалізація його роботи на 3D-сцені за допомогою популярного набору API (Application Programming Interface) DirectX11, використовуючи матеріал та проведені дослідження в області комп’ютерної графіки різними науковцями. SSAO

повинен працювати водночас і швидко, і коректно, і надавати правдоподібні результати затінення, для чого потрібно буде використати нестандартні методи та різного роду оптимізації, які будуть наближено відповідати фізично коректному відображенню.

В процесі роботи будуть вирішуватися наступні завдання:

- використовуватись метод Монте-Карло замість взяття інтеграла по півсфері , так як завдяки методу Монте-Карло та деякого хаотичного патерну повороту півсфери можна досягти приблизно такого ж результату, але розрахунки будуть набагато швидшими
- білатеральне розмиття по Гаусу (bilateral Gaussian blur) для «згладжування» результату, беручи до уваги як наближеність кожного фрагменту один до одного, так і різницю в їх глибинах.
- метод затінення по Блін-Фонгу для швидкого та простого затінення сцени, симулюючи поведінку світла, враховуючи напрямки світла, нормаль до поверхні, напрямок від поверхні до спостерігача, тощо
- Deferred rendering, тобто розрахунок поведінки світла буде відбуватись окремим етапом в екранному просторі (screen space), використовуючи попередньо згенеровані текстури з даними про нормалі, позиції та колір об’єкту. Так як розрахунок SSAO також відбувається в екранному просторі, deferred rendering вписується набагато краще, ніж стандартний forward rendering.

Усе це є апроксимацією та не є фізично коректним, але результат повинен бути наближеним до реальної поведінки світла, так як усі розрахунки будуть використовуватись в рендерингу в реальному часі, а тому вони повинні бути швидкими та вписуватись у 60fps (60 кадрів в секунду).

SSAO – техніка, що призвана поліпшити зображення, додавши до нього багато деталей за рахунок нескладних та, найголовніше, швидких обчислень.

Візуально SSAO виглядає начебто до зображення додали тіні, хоча це не так. Це дуже важливо, адже на правильне обчислення тіней потрібно набагато більше часу, ніж на обчислення SSAO. Звісно, існують інші техніки ambient occlusion екранного простору , такі як SSDO, HBAO, HBAO+, VXAO.

SSDO – Screen Space Directional Occlusion, техніка ambient occlusion екранного простору, що лише нещодавно почала використовуватись. Її особливість в тому, що завдяки SSDO-алгоритму можна симулювати одне відбиття світла (bounce) від поверхні, що необхідно для техніки затемнення GI (Global Illumination). Саме тому SSDO використовують в основному тоді, коли головний алгоритм освітлення сцени – це Global Illumination.

HBAO та HBAO+ – Horizontal Based Ambient Occlusion, техніка ambient occlusion екранного простору, що була представлена компанією Nvidia. Головна її особливість в тому, що в алгоритмі затемнення також приймає участь буфер глибини (depth buffer), що дозволяє більш якісно обробити зображення, але й потребує більше ресурсів.

VXAO – Voxel Accelerated Ambient Occlusion – також алгоритм затемнення від компанії Nvidia, особливість в тому, що до уваги береться не просто 2D піксель, а 3D-voxel (воксель).

Усі техніки ambient occlusion екранного простору працюють на основі однієї ідеї, яку видозмінюють в залежності від очікуваних результатів та доступних ресурсів.

Робота виконувалась по аналогії з роботами Joey de Vries, John Chapman, використовувалась література з презентацій та статей компанії Nvidia.

1 ТЕОРІЯ СВІТЛА ТА ЗАТІНЕННЯ В КОМП’ЮТЕРНІЙ ГРАФІЦІ

1.1 Загальні положення теорії світла для комп’ютерної графіки

Одна із основних проблем комп’ютерної графіки це вирішення так званого rendering equation, або « рівняння рендеру » (надалі – rendering equation). В комп’ютерній графіці rendering equation – це інтегральне рівняння, яке визначає кількість світлового випромінювання у певному напрямку як суму власного та відбитого випромінювань в наближенні геометричної оптики. Рівняння вперше було опубліковано одночасно в роботах Девіда Іммела (David Immel) та Джеймса Каджиї (James Kajiya) в 1986 р. Велика кількість алгоритмів комп'ютерної графіки розв'язують це основне рівняння.

Фізичною основою рівняння є закон збереження енергії. Нехай L — це кількість випромінювання в заданому напрямку у заданій точці простору. Тоді кількість вихідного випромінювання (L_o) — це сума випроміненого (L_e) і відбитого світла. Відбите світло може бути поданим у вигляді суми випромінювання (L_i), що приходить по всім напрямкам, помноженого на коефіцієнт відбиття з даного кута.

Рівняння рендеру має наступний вигляд (1.1) :

$$L_o(x, \omega_0, \lambda, t) = L_e(x, \omega_0, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_0, \lambda, t)\, L_i(x, \omega_i, \lambda, t)(\omega_i * n)\, d\omega_i$$

(1.1)

де:

- x – задана точка у просторі
- $\omega_0 -$ напрямок вихідного світла
- $\lambda -$ довжина хвилі світла
- $t -$ час
- $L_o(x, \omega_0, \lambda, t)$ – це кількість вихідного випромінювання (spectral radiance) с довжиною хвилі λ, спрямованого вздовж напрямку ω_0 в час t з заданої точки x
- $L_e(x, \omega_0, \lambda, t) -$ випромінюване світло
- $n -$ нормаль до поверхні в точці x
- $\omega_i -$ зворотний напрямок падаючого світла
- $\Omega -$ одинарна напівсфера з центром в точці x навколо нормалі n, що містить усі можливі значення ω_i
- $f_r(x, \omega_i, \omega_0, \lambda, t) -$ bidirectional reflectance distribution function (BRDF - двонаправлена функція розподілу відбиття), кількість (пропорція) випромінювання, відбитого з напрямку ω_i у напрямку ω_0 в точці x, в час t на довжині хвилі λ
- $L_i(x, \omega_i, \lambda, t) -$ кількість випромінювання з довжиною хвилі λ що надходить до точки x з напрямку ω_i.
- $\omega_i * n -$ коефіцієнт ослаблення отриманого випромінювання по заданому куту між напрямками ω_i та n. Часто записується як $\cos \Theta_i$

Spectral radiance (енергетична яскравість) - енергетична фотометрична величина, відношення потоку випромінювання, що випускається з нескінченно малої площинки джерела і поширюється в нескінченно малому тілесному куті, до площі проекції цієї площинки на площину, перпендикулярну до напрямку поширення, і величини тілесного кута. Іншими словами це відношення потоку випромінювання на одиницю тілесного кута на одиницю площі проектованої поверхні.

Bidirectional reflectance distribution function (двонаправлена функція розподілу відбиття) - це функція чотирьох дійсних змінних, яка визначає, як світло відбивається від непрозорої поверхні. Функція приймає напрямок падаючого світла ω_i, та напрямок відбитого світла ω_r і повертає відношення відбитого випромінювання (radiance) світла уздовж напрямку ω_i до опромінення (irradiance) падаючого на поверхню світла у напрямку ω_r.

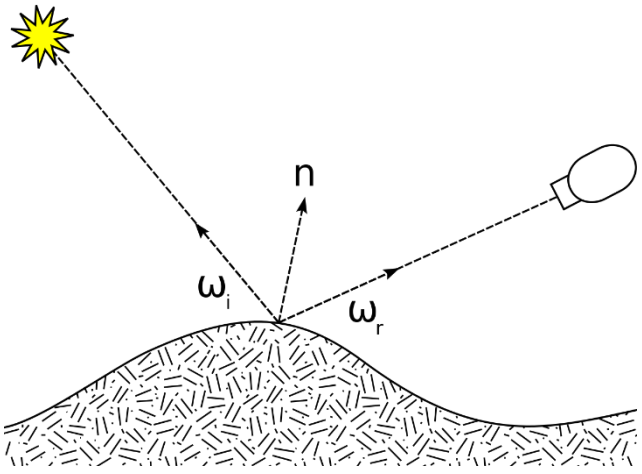


Рисунок 1.1.1 – фізичний зміст BRDF

1.2 Апроксимація рівняння рендеру для real-time rendering

Одною з основних задач комп’ютерної графіки є вирішення рівняння рендеру для кожної точки в просторі, відбите світло з яких може потрапити в око спостерігача. Нажаль, потужності сучасних відеокарт та процесорів недостатньо для того, щоб можна було в реальному часі фізично достовірно обчислити відбиття світла навіть від одного джерела світла. Саме цьому для real-time рендерингу існує багато різних апроксимацій поведінки світла в просторі, які можуть бути не на 100% фізично обґрунтованими, але надають можливість приблизно відтворити поведінку світла від різних поверхонь за різних умов.

Shading model (модель затінення) – модель розрахунку затінення поверхонь 3D-об’єктів, в тому числі полігональних моделей та примітивів (частіше всього трикутників) а також метод інтерполяції затінення по всьому об’єкту.

Існує багато різних моделей затінення, всі вони відрізняються методом розрахунку затінення 3D-об’єкта. Деякі більш прості і майже зовсім не відтворюють фізичну поведінку світла, інші намагаються якомога точніше передати фізичну поведінку світла, але вони й більш складні та вимагають більше часу (і більшу потужність) для розрахунку кожної точки в просторі. Нижче наведені найбільш популярні та швидкі локальні моделі затінення для realtime rendering:

- Lambertian reflectance
- Phong shading model
- Blinn-phong shading model

1.2.1 Lambertian reflectance

Lambertian reflectance (ламбертове відбиття) – це властивість, що визначає ідеальну «матову» поверхню або ідеальне дифузне (розсіяне) відбиття світла. Дифузне розсіювання – таке розсіювання, при якому світлові промені відбиваються в різних напрямках. Для спостерігача явна яскравість ламбертового відбивання є однаковою незалежно від кута зору спостерігача. Більш технічно, яскравість поверхні є ізотропною, а інтенсивність освітлення підкоряється закону Ламберта. Ламбертове відбивання назване на честь Йогана Генріха Ламберта, який представив концепцію ідеальної дифузії у своїй книзі «Фотометрія» 1760 року.

Інтенсивність світла відбитого точкою залежить лише від напрямку нормалі поверхні у цій точці та напрямку падаючого світла. Вона не змінюється при обертанні вектора нормалі поверхні навколо вектора падаючого світла. Інтенсивність відбиття розраховується як скалярний добуток вектора нормалі поверхні n і вектора напрямку світла ω_i, що спрямований від поверхні до точкового джерела світла. Отриманий результат множиться на альбедо поверхні C та на інтенсивність світла I_l, що потрапляє на поверхню :

$$I_d = \omega_i * n * C * I_l \qquad (1.2)$$

1.2.2 Phong shading model

Phong shading model (Модель затінення по Фонгу) (рисунок 1.2.2.2) – більш складна апроксимація відбиття світла від поверхні. В основні моделі лежить три компоненти:

- Ambient lighting (навколишнє освітлення)
- Diffuse lighting (дифузне освітлення)

- Specular lighting (дзеркальне освітлення)

Ambient lighting. Об’єкти майже ніколи не бувають повністю чорними, тому що майже завжди є якесь далеке джерело світла, промені з якого доходять до поверхні. Тому за допомогою ambient lighting симулюють таку поведінку світла, нібито якесь інше світло таки потрапило на поверхню.

Diffuse lighting. Дифузне розсіювання, для якого обов’язково потрібно знати нормаль до поверхні та напрямок світла.

Specular lighting. Дзеркальне розсіювання світла, при якому визначається дзеркально відбитий промінь від джерела світла в точці розрахунку освітлення. (рисунок 1.2.2.1). Дзеркальне відбиття у моделі Фонга симулює яскраву «точку» світла, яке проявляється на блискучих об’єктах. (рисунок 1.2.2.2). В розрахунку дзеркального розсіювання окрім нормалі до поверхні та напрямку світла також приймає участь вектор напрямку з точки освітлення до ока спостерігача. (рисунок 1.2.2.3)

При поєднанні всіх вищезазначених компонентів зображення набуває вигляду реального освітленого об’єкту, де важливе як позиція джерела світла, так і позиція спостерігача.

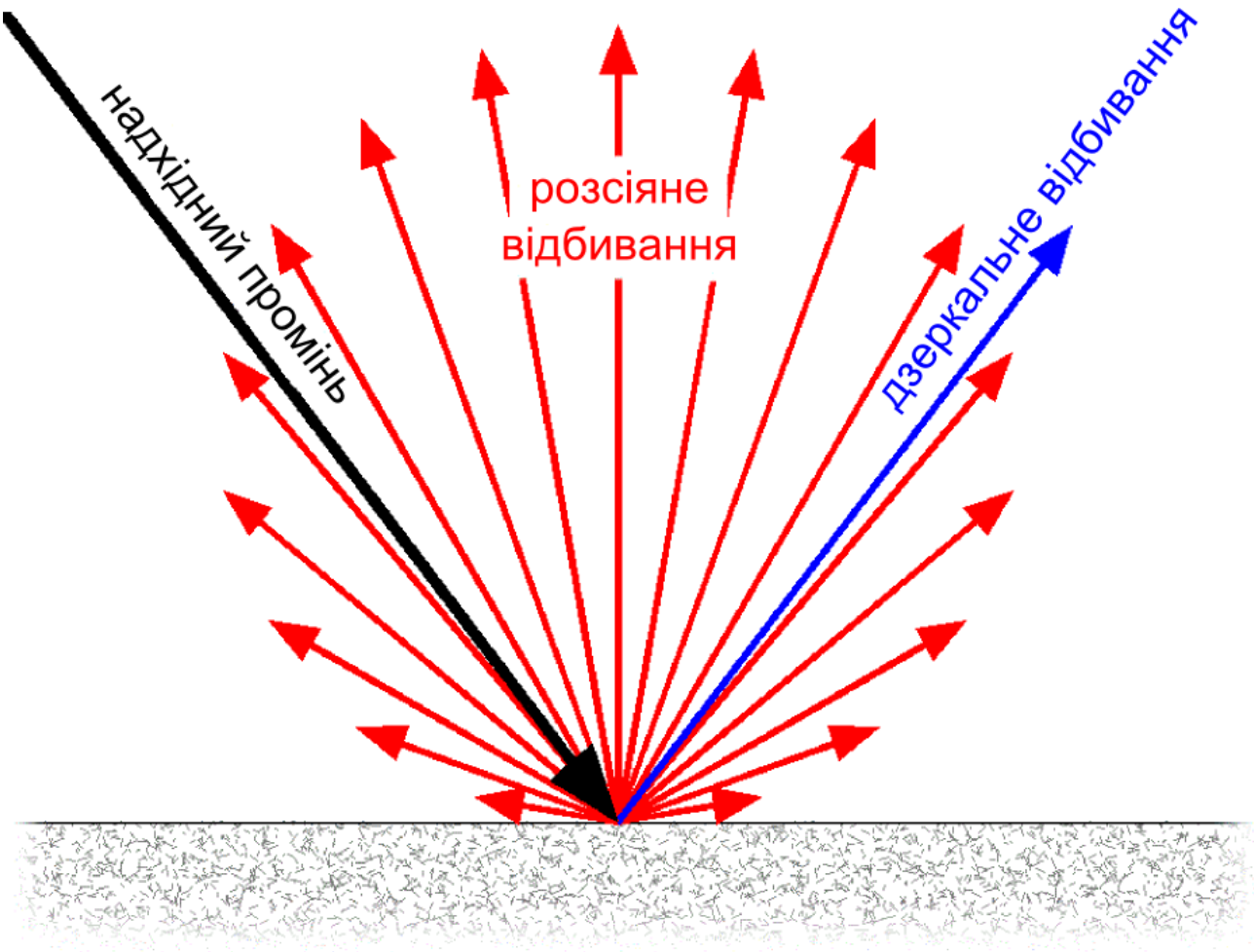


Рисунок 1.2.2.1 Дифузне та дзеркальне розсіювання світла

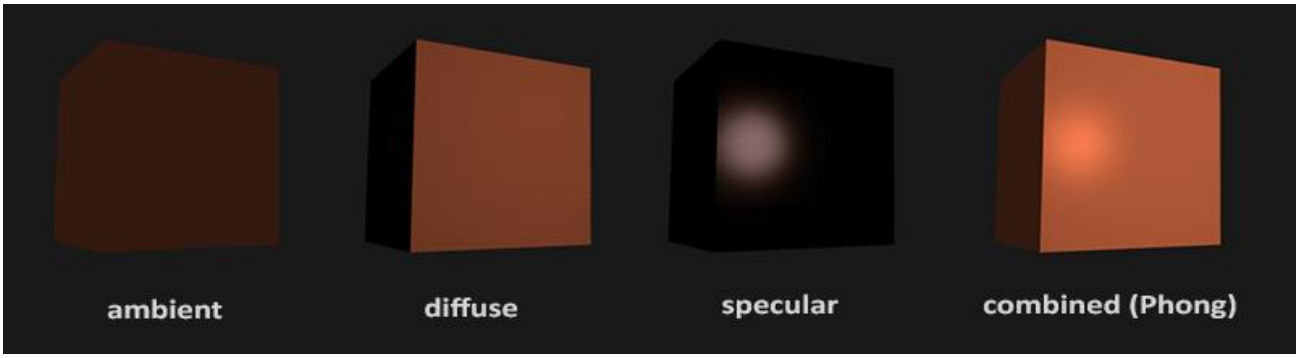


Рисунок 1.2.2.2 Модель затінення Фонга та всі її компоненти

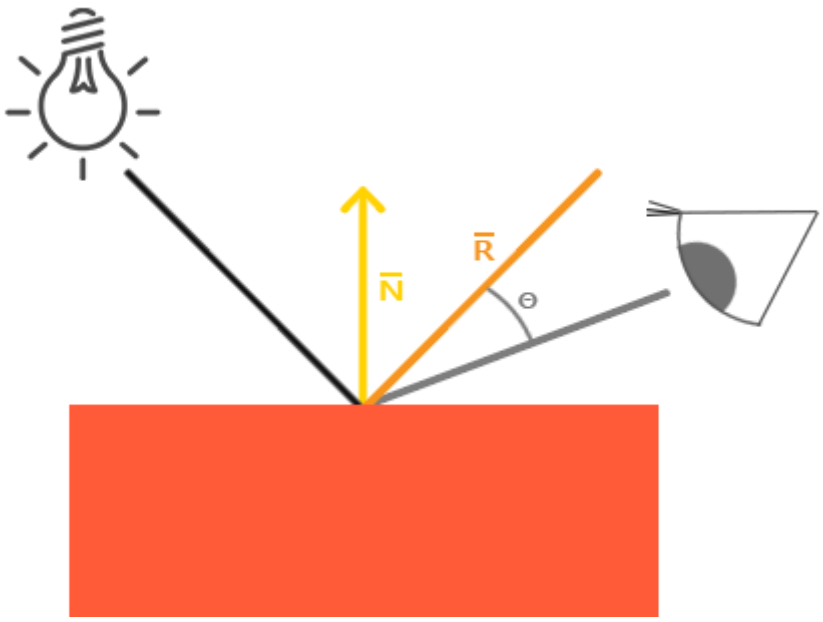


Рисунок 1.2.2.3. Specular lighting для моделі затінення по Фонгу

Загальна формула для моделі затінення по Фонгу виглядає наступним чином:

$$L_{ph} = L_a + L_d + L_s \qquad (1.3)$$

, де:

- L_{ph} - колір фрагменту по моделі Фонга
- L_a- ambient освітлення
- L_d- diffuse освітлення
- L_s- specular освітлення

L_a- ambient компонента зазвичай дорівнює альбедо об’єкта, тобто його кольору, тобто

$$L_a = C * C_l * A \quad (1.4)$$

де:

- C – константний колір об’єкту
- $C_l -$ колір світла
- $A -$ фактор для зменшення впливу ambient освітлення на об’єкт

$$L_d = \omega_i * n * C * I_l \quad (1.5)$$

де:

- $\omega_i -$ напрямок світла, що спрямований від поверхні до точкового джерела світла.
- $n -$ нормаль до поверхності
- C – константний колір об’єкту
- $I_l -$ інтенсивність світла

$$L_s = (V * R)^B * I_l * C_l \quad (1.6)$$

де:

- $V -$ вектор напрямку від точки освітлення до спостерігача (ока)
- $R -$ вектор напрямку дзеркально відбитого промені світла в точці освітлення навколо нормалі до поверхні
- $n -$ нормаль до поверхності
- $C_l -$ колір світла
- $I_l -$ інтенсивність світла
- $B -$ фактор, який відображає наскільки багато світла дзеркально відображається від поверхні

1.2.3 Blinn-Phong shading model

Blinn-Phong shading model (модель затінення Блін-Фонга) – доповнення до моделі затінення Фонга, що дозволяє завдяки апроксимації полегшити обчислення майже без змін у візуальному вигляді освітлення.

Основна ідея в тому, що для для компонента Specular в моделі затінення Фонга потрібно знайти вектор, що відображається від вектора напрямку світла навколо нормалі до поверхні. Для того, щоб знайти відображальний вектор R, потрібно виконати декілька «важких» операцій, що може здійснюватись досить довго на менш потужних відеокартах та (або) процесорах. Науковець Джим Блін (Jim Blinn) визначив, що замість відношення відбитого променю світла R до вектора напрямку від освітлюваної точки до ока V, можна використовувати відношення так званого half-вектору H до нормалі поверхні N. (рисунок 1.2.3.1)

Звісно, такий спосіб є менш фізично коректним, але надає правдоподібні результати за менший проміжок часу, тому він де-факто замінив модель затінення по Фонгу.

Саме модель затінення по Блін-Фонгу найчастіше використовується у різного роду візуалізаторах, де не потрібно затінювати зображення фізично коректно, але потрібно щоб затінення відбувалось швидко.

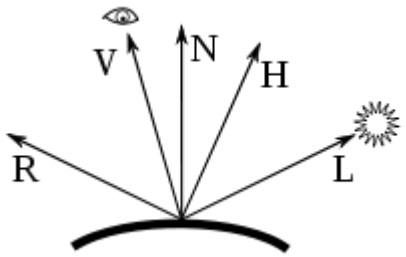


Рисунок 1.2.3.1 Фізичний зміст моделі Blinn-Phong

В такому разі, замість кута між R та V використовується кут між H та N, де

- L – вектор напрямку від точки до джерела світла
- N – нормаль до поверхні
- V – вектор напрямку від точки до ока спостерігача
- R – вектор відбиття зворотнього вектору L навколо нормалі N
- H – half-вектор, що визначається як $H = \frac{L+V}{||L+V||}$

Отже, формула для Specular компоненти в моделі затінення Blinn-Phong має наступний вигляд:

$$L_s = (H * n)^B * I_l * C_l \qquad (1.7)$$

, де :

- H –half-вектор
- $n -$ нормаль до поверхності
- $C_l -$ колір світла
- $I_l -$ інтенсивність світла
- $B -$ фактор, який показує наскільки багато світла дзеркально відображається від поверхні

Дослідним шляхом було виявлено, що модель затінення Blinn-Phong візуальне навіть краще, ніж модель Phong, хоча вона й менш фізично-коректна.

У дипломній роботі буде використовуватись саме модель затінення Blinn-Phong, адже це більш-менш коректний та найшвидший спосіб затінити зображувані 3D-об’єкти.

Модель затінення Blinn-Phong є локальною, тобто яскравість (освітленість) точки на поверхні не залежить від її оточення та іншим об’єктів на сцені, що не є фізично коректним, але дозволяє швидко досягти непоганих результатів по затіненню поверхні.

1.3 Ambient Occlusion

Ambient Occlusion – модель затінення, що використовується в комп’ютерній графіці та дозволяє додати реалістичності зображенню за допомогою обчислення інтенсивності світла, що надходить до поверхні. На відміну від локальних методів затінення (як, наприклад, модель затінення Blinn-Phong), ambient occlusion являється глобальним методом, тобто значення функції залежить від інших об’єктів на сцені.

В основі ambient occlusion лежить ідея, що світло має меншу інтенсивність на поверхнях, які не можуть отримати світло з усіх напрямків. Наприклад, в кутках кімнати, дірах, на поверхнях, що знаходяться близько один до одного. Усі ці ділянки значною мірою закриті (звідси і «occlusion») геометрією навколишнього середовища, а отже світлові промені мають менше місць, звідки можна потрапити на поверхню і відбитись від неї, тому такі області і поверхні виглядають більш затемненими. Загалом, ambient occlusion являється апроксимацією поведінки «непрямого» світла (indirect lighting), так як прорахувати поведінку кожного променю світла неможливо.

Ambient occlusion зараз використовується в будь-якому медіа, де потрібно затінити 3D-об’єкти на сцені. Це і фільми, і ігри, різного роду візуалізатори, віртуальна реальність (VR) тощо. Для віртуальної реальності потрібно також брати до уваги той факт, що зображення потрібно рендерити два рази (для кожного «ока»), тому SSAO є одною з найважливіших складових рендерингу для VR, адже це швидкий спосіб поліпшити зображення.

Ambient occlusion найчастіше обчислюється шляхом побудови променів, що виходять з точки поверхні у всіх напрямках, з подальшою їх перевіркою на перетин з іншими об'єктами на сцені. Промені, що досягли фону (тобто не перетнулись з іншими об’єктами на сцені), збільшують яскравість поверхні, в той час як промені, які перетинають інші об'єкти, зменшують яскравість. В результаті точки, оточені великою кількістю геометрії, візуалізуються як більш темні, а точки з малою кількістю геометрії навколо - світлими.

Ambient occlusion це досить «дорога» в плані ресурсів комп’ютера техніка, так як потрібно брати в розрахунок оточуючу геометрію для кожної точки, що затінюється. Гіпотетично, можна запустити велику кількість променів напрямком від точки на поверхні до кожної точки в просторі, дивитися скільки з цих променів досягли поверхні, і таким чином зрозуміти кількість затемнення, що потрібно для даної точки на поверхні, але це неможливо в реаліях потужності сьогоденних процесорів та відеокарт.

Для ambient occlusion використовуються багато різних технік, які відрізняються як складністю розрахунку, так і візуальним результатом.

Одним із найефективніших прийомів для ambient occlusion є ambient occlusion map. Ambient occlusion map – це окрема текстура для об’єкта, в якій записані уже попередньо оброблені значення для ambient occlusion, тому в реальному часі, коли цей об’єкт затемнюється, потрібно просто зчитати правильне значення ambient occlusion-у з текстури і використовувати його для

затемнення об’єкту. Це швидше та візуально краще, ніж вираховувати значення ambient occlusion-у в реальному часі самостійно, але потребує попередньої підготовки текстури для об’єкта, тому цей спосіб зазвичай використовується тільки для дійсно значущих та складних об’єктів, де дуже важливо відтворити максимально реалістично затемнення об’єкту. Для ігор це зазвичай персонажі та об’єкти, що завжди знаходяться у фокусі камери.

2 SSAO – SCREEN SPACE AMBIENT OCCLUSION

2.1 Загальні положення SSAO

В 2007 році компанія Crytek опублікувала статтю, де описала нову техніку, яку назвала SSAO (Screen Space Ambient Occlusion), або ambient occlusion екранного простору. Цю техніку компанія використовувала у її грі Crysis, що довго вважалась еталоном real-time комп’ютерної графіки. Техніка використовує depth buffer (буфер глибини) в екранному просторі для того, щоб визначити ambient occlusion фактор для кожної точки на екрані. Це набагато ефективніше та швидше, ніж використовувати реальні геометричні дані кожного об’єкту на сцені. Цей підхід неймовірно швидкий, порівнюючи з реальним геометричним ambient occlusion-ом, та дає правдоподібні результати, тому став де-факто основним для апроксимації real-time ambient occlusion.

Основний принцип SSAO досить простий: для кожного фрагменту на екрані (тобто пікселю) потрібно вирахувати occlusion фактор, тобто наскільки сильно потрібно затемнити фрагмент, базуючись на значеннях в depth buffer (буфері глибини) сусідніх фрагментів. Occlusion фактор потім використовується

для затемнення фрагменту. Occlusion фактор отримують, зчитуючи декілька значень із depth buffer-а в деякому радіусі від фрагмента, що розглядається, і потім ці значення порівнюються зі значенням глибини основного фрагменту. Кількість глибинних значень, що більше за глибину фрагменту, що розглядається, і являються occlusion фактором.

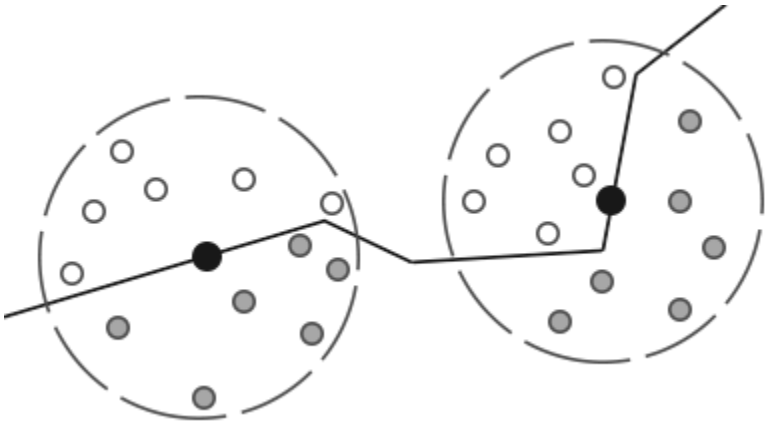


Рисунок 2.1.2. Сфери, в радіусі яких беруться сусідні з фрагментом точки та зчитується їх значення глибини. Сірі – точки, глибини яких більша за глибину фрагмента, білі – навпаки.

Чим більше точок, глибина яких більше за глибину фрагменту, що розглядається, тим більше повинен бути occlusion фактор, тим темніше повинен бути фрагмент.

Дана апроксимація не є фізично коректною, але вона допомагає досягти правдоподібних результатів при розрахунку поведінки світла в місцях дотику різних об’єктів, адже глибини об’єктів будуть різнитися досить часто, якщо обробляється фрагмент на стику двох різних об’єктів (мається на увазі геометрія сцени, об’єкт насправді може бути один і той же, головне те, що до

даного фрагменту світло не зможе потрапити з усіх сторін, так як він перекривається іншими об’єктами).

2.2. Реалізація SSAO

Перш за все, замість того, щоб зчитувати глибину в радіусі сфери навколо фрагменту, в даній дипломній буде зчитуватись глибина в радіусі півсфери, орієнтованої по нормалі до поверхні. (рисунок 2.2.1). Такий підхід також був опублікований, і він дає більш реалістичний результат, а так як в дипломній роботі використовується deferred rendering, то і не набагато повільніше в реалізації.

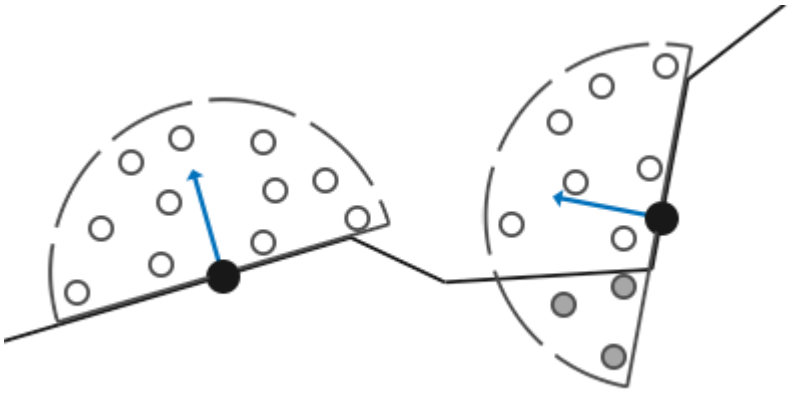


Рисунок 2.2.1. Півсфери, замість сфер, для зчитування глибинного буферу

Якщо точки будуть зчитуватися по півсфері, а не сфері, то фінальний occlusion фактор буде меншим, так як до уваги вже не беруться точки, глибина яких була 100% більшою за глибину фрагменту, що розглядається. (сірі точки першої сфери на рисунку 2.1.2). Тому occlusion фактор для фрагментів, біля яких немає інших об’єктів, буде дорівнювати нулю, тобто він не буде враховуватися при затемненні фрагменту.

Для правдоподібного результату кількість точок у півсфері повинна бути дуже великою, що погано позначиться на швидкості обробки результатів. John Chapman (Джон Чапман) запропонував додати випадкові числа (рандом) для розрахунку occlusion фактору. Я буду використовувати схожий рандом. Якщо мати, наприклад, 32 випадкових точки всередині півсфери, та хаотично обертати їх навколо нормалі, можна отримати досить правдоподібний результат без впливу на швидкість обробки результатів. Нажаль, це буде помітно на фінальній картинці, так як такий noise pattern (шумний патерн картинки) дуже виділяється. Але це можна вирішити, якщо після обрахунку occlusion фактору для кожного фрагменту застосувати blur (розмиття). Так як даний спосіб передбачає high-frequency noise (високочастотний шум), то Gaussian blur (розмиття по Гаусу) буде ідеальним вибором, щоб представити фінальну картинку.

Отже, для розрахунку SSAO потрібні наступні дані:

- Вектор позиції фрагменту в view-space (простір камери)
- Вектор нормалі до поверхності фрагменту в view-space
- Альбедо колір фрагменту
- Масив точок всередині півсфери (sample kernel)
- Вектор хаотичного повороту для того, щоб повернути sample kernel навколо нормалі

В дипломній роботі використовується deferred rendering, що означає після того, як намалювалась геометрія, в доступності є декілька текстур, заповнених різними даними, що потрібні для SSAO та подальшого затемнення фрагментів.

2.2.1 Текстура векторів позицій

Для запису вектору позиції фрагменту в текстуру я використовую позицію кожного фрагменту в model-space (простір моделі), потім перевожу в систему координат світу (world-space), потім перевожу в систему координат камери (view-space).

Я використовую матриці на основі стовпців, отже фінальна формула має вигляд:

$$P_v = M_v * M_w * P_m \quad (2.1)$$

Де:

- $P_v -$ фінальна позиція в системі координат камери (view – space)
- $M_v -$ матриця переходу з системи координат світу в систему координат камери (world space -> view space)
- $M_w -$матриця переходу з системи координат моделі в систему координат світу (model local space -> world space)
- $P_m -$координати точки в системі координат моделі

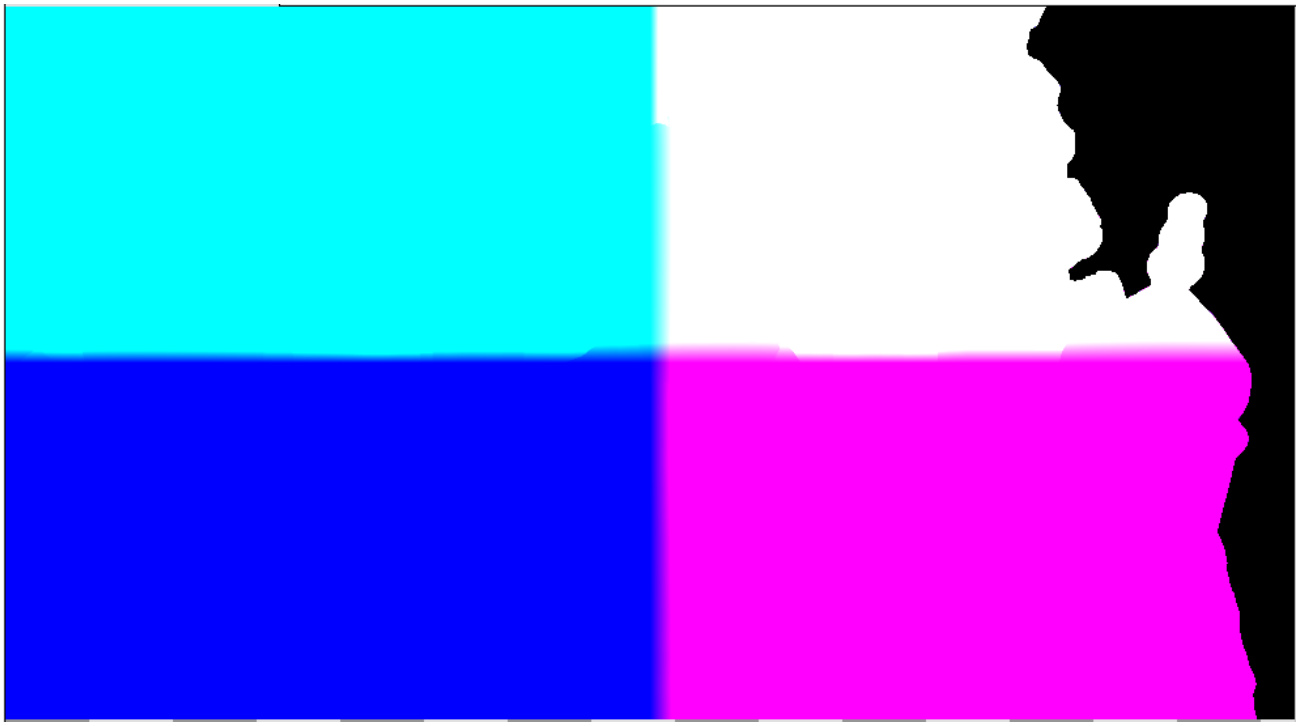


Рисунок 2.2.1.1 Текстура, де кожний фрагмент (піксель) має значення позиції в системі координат камери

2.2.2 Текстура векторів нормалей

Для запису вектору нормалі в текстуру я також використовую нормаль кожного фрагменту в model-space (простір моделі), потім перевожу в систему координат світу (world-space), потім перевожу в систему координат камери (view-space).

Для того, щоб перевести нормаль с одної координатної системи в іншу, потрібно використовувати матрицю нормалей.

Матриця нормалей – це матриця, яка переводить нормаль з однієї системи координат в іншу. Вона особлива, тому що нормаль – це нормалізований напрямок, тому для нього не можна застосовувати ні переміщення (translation), ні масштабування (scale). Тому для матриці нормалей використовується інверсивна транспонована обрізана до 3x3 матриця переходу с системи координат моделі в систему координат камери.

Матриця обрізається з розмірності 4x4 до 3x3 для того, щоб при перемноженні не використовувати переміщення. А інверсивна і транспонована вона для того, щоб не використовувати масштабування. До речі, якщо використовується тільки юніформне масштабування (однакове по всім трьом компонентам $X\ Y\ Z$), то можна використовувати просто матрицю 3x3.

Формула матриці нормалей:

$$M_n = (3x3)inverse(transpose((M_v * M_w))) \quad (2.2)$$

, де:

- $M_n -$ матриця нормалей. переходу з системи координат моделі в систему координат камери (model local space -> view space)
- $M_v -$ матриця переходу з системи координат світу в систему координат камери (world space -> view space)
- $M_w -$матриця переходу з системи координат моделі в систему координат світу (model local space -> world space)

Фінальна формула для обчислення нормалі в системі координат камери має вигляд:

$$N_v = M_n\ * N_m \quad (2.3)$$

Де:

- $N_v -$ фінальний вектор нормалі в системі координат камери (view – space)
- $M_n -$ матриця нормалей. переходу з системи координат моделі в систему координат камери (model local space -> view space)
- $P_m -$координати точки в системі координат моделі

Рисунок 2.2.2.1 Текстура, де кожний фрагмент (піксель) має значення нормалі в системі координат камери

Необхідно сказати, що досить часто для deferred rendering нормалі та позиції записують в системі координат світу (world-space). В моєму випадку, усі розрахунки світла проводяться в системі координат камери (view space), тому буде набагато продуктивніше одразу зберігати дані компоненти в системі координат камери, а не переводити їх із одного базису в інший, адже ці обчислення також займають багато часу.

2.2.3 Текстура векторів альбедо

Альбедо колір об’єкту записати в окрему текстуру досить просто, так як не потрібно переводити нічого в другу систему координат, так що просто можна записати константний колір об’єкту в текстуру. (рисунок 2.2.3.1). Наразі це константний колір для кожного об’єкту, але його можна змінювати як для цілого об’єкту, так і для кожної частини об’єкту, використовуючи diffuse color texture.

Рисунок 2.2.3.1 Текстура, де записаний альбедо колір кожного фрагменту

2.2.3 Sample kernel

Замість інтегрування по всій півсфері, що є дуже повільною операцією, можна використати метод Монте-Карло, тобто в даному випадку хаотично згенерувати деяку кількість точок всередині півсфери, орієнтованої по нормалі до поверхні. Це дуже складно і не правдоподібно генерувати такі точки (надалі sample kernel) для кожної нормалі на поверхні, тому є сенс згенерувати точки в tangent space (дотичний простір). Tangent space – це такий простір, де нормаль до поверхні завжди вказує по напрямку z-координати. (дивитись рисунок 2.2.3.1)

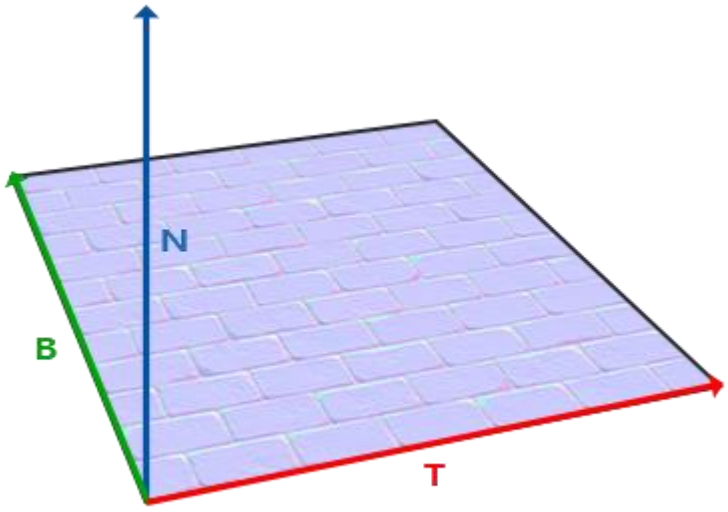


Рисунок 2.2.3.1. Tangent space. N це нормаль, вказує по напрямку z-координати. B – бітангент, вказує по напрямку y-координати, T- тангент, вказує по напрямку x-координати.

Потрібно хаотично згенерувати декілька точок в дотичній системі координат (tangent space), що будуть всередині одиничної півсфери (півсфери з радіусом 1), яка зорієнтована навколо нормалі до поверхні. Отже, координата x та y повинні лежати в межах $[-1;\ 1]$, а координата z в межах $[0;\ 1]$. Також потрібно щоб точок було більше до центру напівсфери, і менше на кінцях радіусу. Отже, фінальна формула для обчислення кожної точки:

$$P_s = normalize(P_r(random(-1,1), random(-1,1), random(0,1))) * random(0,1) * S \quad (2.4)$$

, де:

- $P_s -$ фінальна точка в дотичній системі координат (tangent space)
- $S -$ масштабування, яке забезпечує більшу розподіленість точок біля центру півсфери

$$S = linear\_interpolation(0.1, 1, (\frac{i}{size})) \quad (2.5)$$

, де

- i – номер точки, що наразі генерується
- $size$ –загальна кількість точок

Таким чином, точки будуть розміщені ближче до центра півсфери (рисунок 2.2.3.2.)

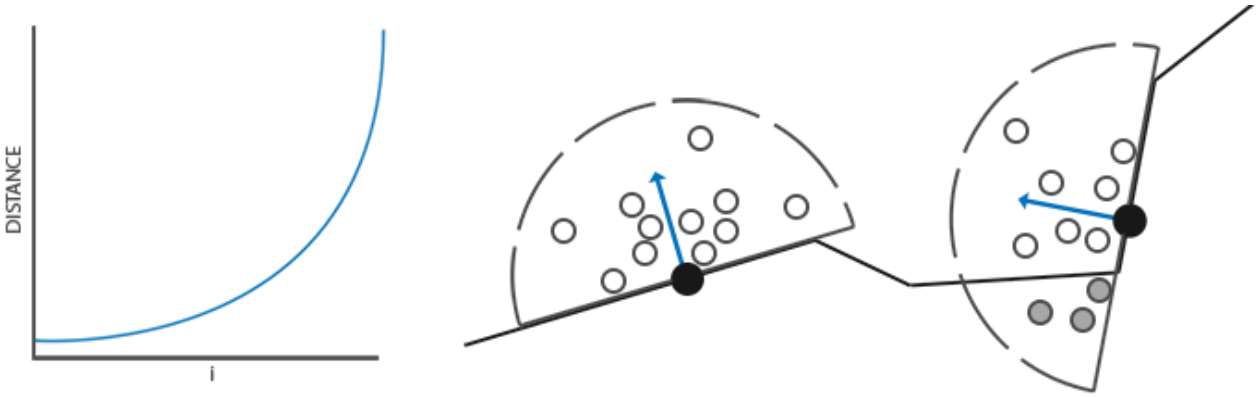


Рисунок 2.2.3.2 Залежність точок від дистанції до центра півсфери

2.2.3 Вектор повороту

Вектори повороту можна записати в текстуру для того, щоб швидко та хаотично отримувати різні вектори повороту для різних фрагментів. z-координата вектору повороту повинна дорівнювати нулю, що означає, що ми повертаєм sample kernel (точки всередині півсфери) навколо z-координати, тобто навколо нормалі до поверхні в дотичній системі координат. x та y координата має лежати в межах $[-1;\ 1]$. На рисунку 2.2.3.1 можна побачити текстуру розмірністю 4x4, в якій записано 16 значень векторів повороту.

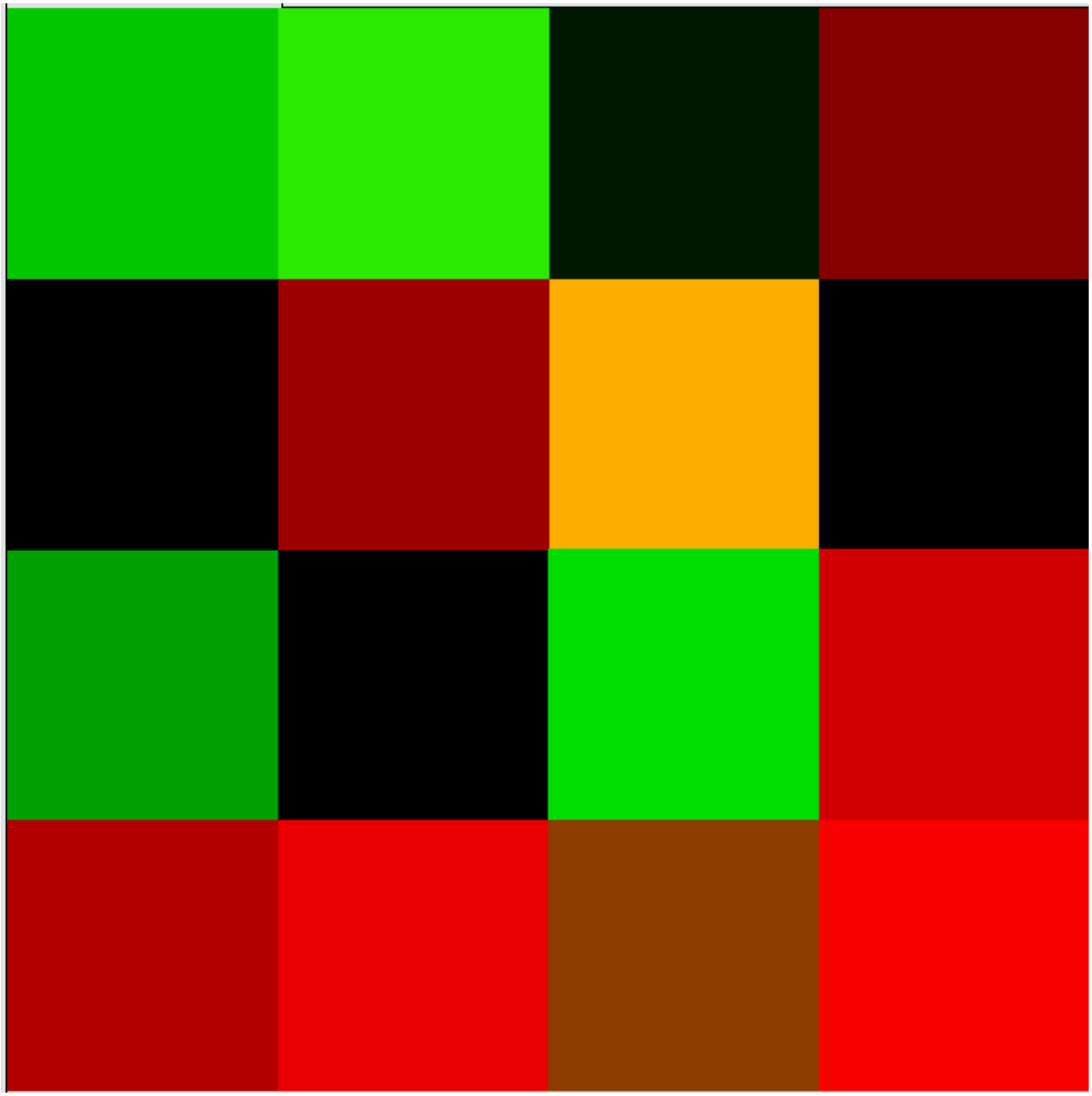


Рисунок 2.2.3.2 Текстура, кожний тексель якої відповідає вектору повороту

2.2.4 Алгоритм визначення occlusion – фактору

Отже, маємо всі дані для розрахунку ambient occlusion фактору. Для кожного фрагменту тепер потрібно зчитати значення позиції, нормалі, вектору повороту, та усі точки sample kernel всередині півсфери (я використовую 32 точки).

Далі необхідно створити матрицю (TBN) переходу із дотичної системи координат в систему координат камери (tangent space -> view space). Для цього нам потрібні три вектори в системі координат камери: нормаль N, тангент T та бітангент B. Нормаль вже відома, так як її можна зчитати з текстури, тангент T знаходиться як:

$$T = normalize(R - N * dot(R, N)) \quad (2.6)$$

, де:

- $T -$ вектор тангент
- $R -$ вектор повороту, що можна зчитати з текстури
- $N -$ вектор нормалі
- $dot(R, N) -$ скалярний добуток R та N

А бітангент можна знайти векторним перемноженням тангенту та нормалі.

$$B = cross(T, N) \quad (2.7)$$

Далі кожну точку с sample kernel потрібно помножити на матрицю TBN, тобто перевести точки з дотичної системи координат в систему координат камери, далі зчитати з кожної з точок вектор позиції з текстури та зрівняти глибину (тобто z-координату) самої точки з глибиною зчитаного вектора позиції. Якщо глибина точки більша, це означає що вона закрита іншим об’єктом на сцені, тобто occlusion-фактор повинен збільшитись.

Також, для того щоб уникнути артефактів при перевірці глибин, потрібно додати невеличкий біас ($bias$), або зміщення, для глибини, так що:

$$Z_t \leq Z_s - bias \quad (2.8)$$

, де:

- $Z_t -$ значення глибини, зчитаної з текстури позицій
- $Z_s -$ значення глибини точки

Також, для того, щоб уникнути затінення тих фрагментів, які знаходяться далеко один від одного по глибині, необхідно додати перевірку на максимум розбіжності в глибинах. Якщо різниця між глибинами занадто велика, це означає що об’єкти знаходяться досить далеко один від одного, тому вони не повинні впливати на occlusion-фактор один одного.

В кінці occlusion-фактор кожного фрагменту записується в текстуру, де три компоненти кольору кожного пікселя дорівнюють $1 - O$, де $O -$ це occlusion фактор в межах $[0; 1]$.

$1 - O$ використовується тому, що колір об’єкту в просторі *RGB* записується для кожного компоненту (red, green, blue) в межах $[0; 1]$, тобто колір зі значенням «0» означає повністю чорний колір (мінмальне значення), а колір зі значенням «1» - максимальний, тобто білий, якщо всі три компоненти (red, green, blue) дорівнюють один одному.

Рисунок 2.2.4.1 Текстура SSAO

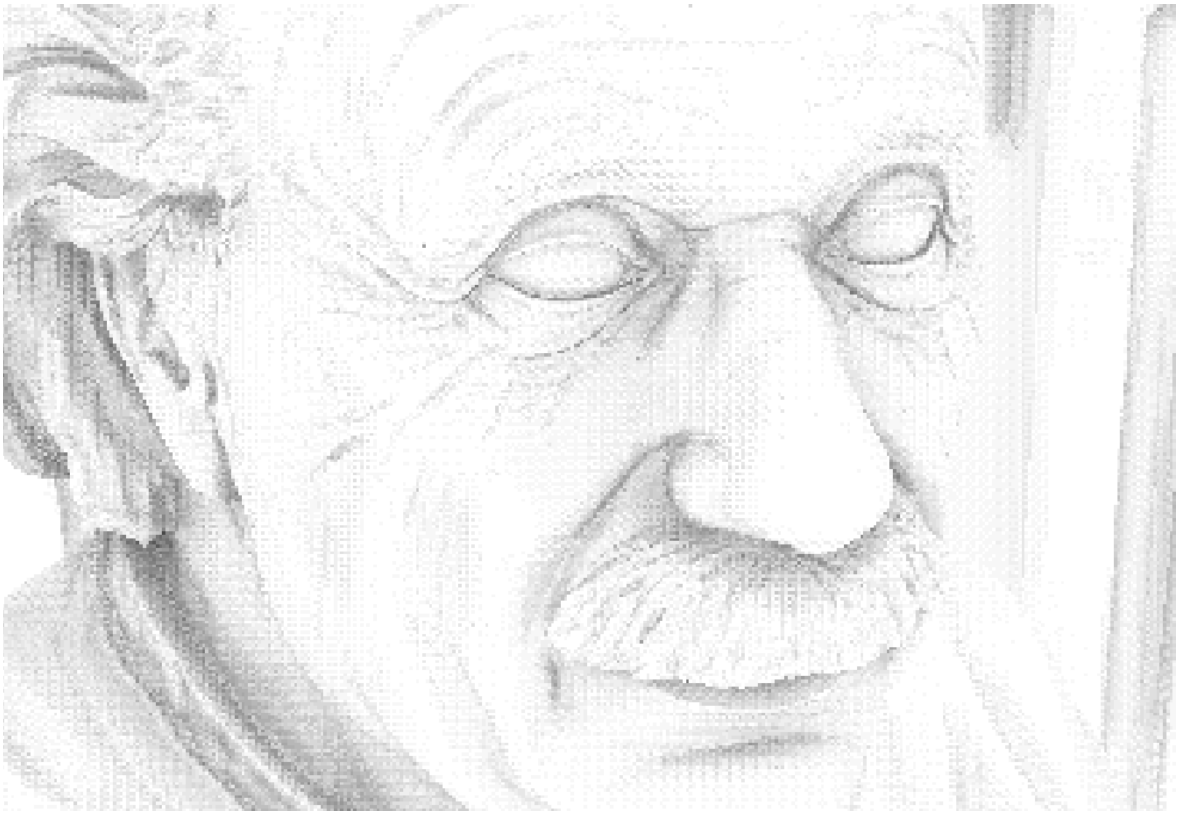


Рисунок 2.2.4.2 Текстура SSAO

На рисунку 2.2.4.2 можна побачити артефакт high-frequency noise, через те, що хаотичні вектори зчитуються для кожного фрагменту і повторюються кожних 16 фрагментів (тому що текстура для хаотичних поворотів розмірності 16x16). Для того, щоб уникнути цього, для SSAO текстури необхідно використати blur (розмиття)

2.2.5 Алгоритм blur

Blur – техніка фільтрації зображення, що призводить до розмиття зображення. Зміст розмиття досить простий: колір кожного фрагменту (пікселю) повинен залежати від кольору сусідніх, оточуючих його фрагментів. На рисунку 2.2.5.1 та 2.2.5.2 зображено розмиття для одного фрагменту, використовуючи тільки одну горизонтальну лінію. На практиці ж використовується двовимірна матриця фрагментів, так як потрібно брати до уваги усі оточуючі фрагменти. Якщо радіус розмиття дорівнює 2-ум, то до уваги потрібно брати матрицю фрагментів 5x5, тобто для визначення кольору одного фрагмента потрібно зчитувати та «змішувати» дані з 25 фрагментів (рисунок 2.2.5.3). Для простого алгоритму blur для кожного фрагмента береться середнє арифметичне усіх сусідніх фрагментів та його самого.

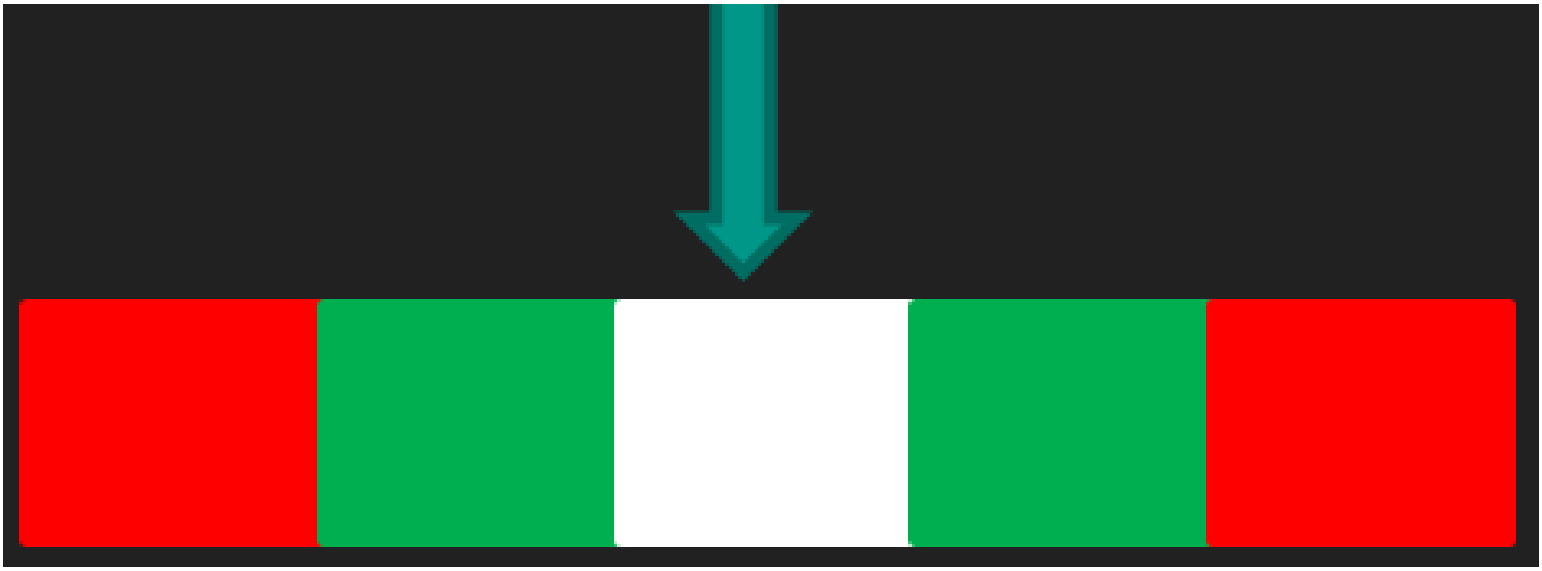


Рисунок 2.2.5.1. П’ять фрагментів, що беруться до уваги з однієї горизонтальної лінії, якщо радіус blur-а дорівнює двум

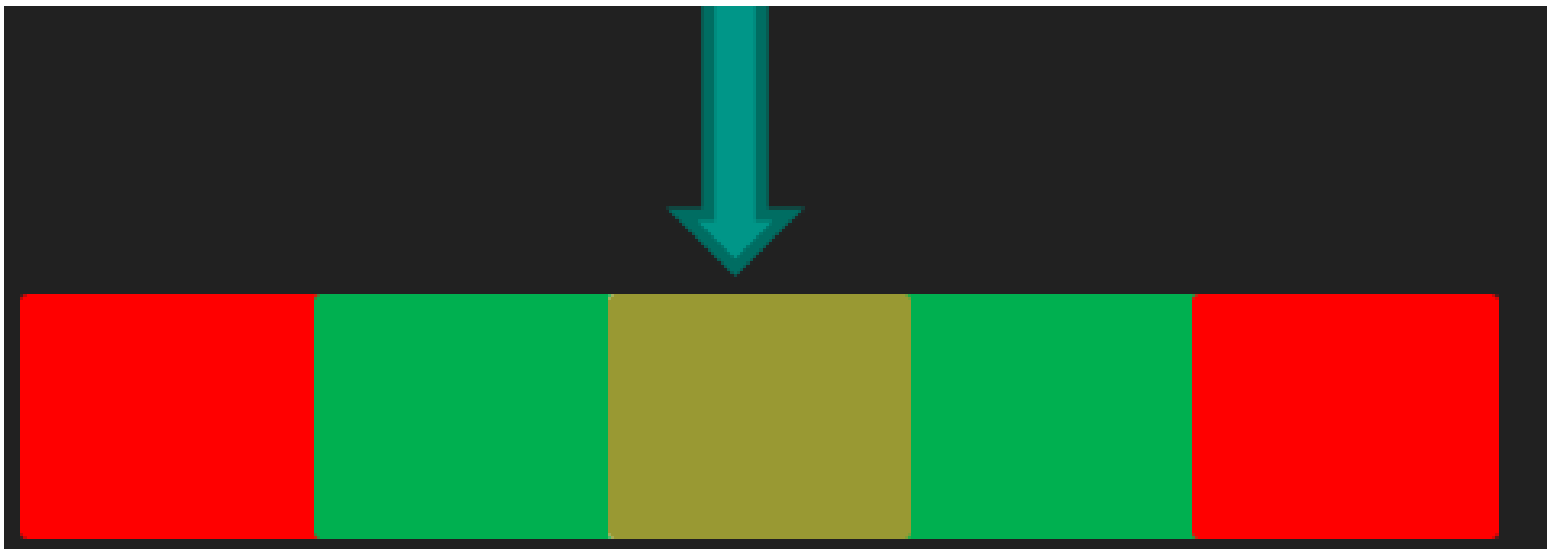


Рисунок 2.2.5.2. Результат розмиття центрального фрагменту

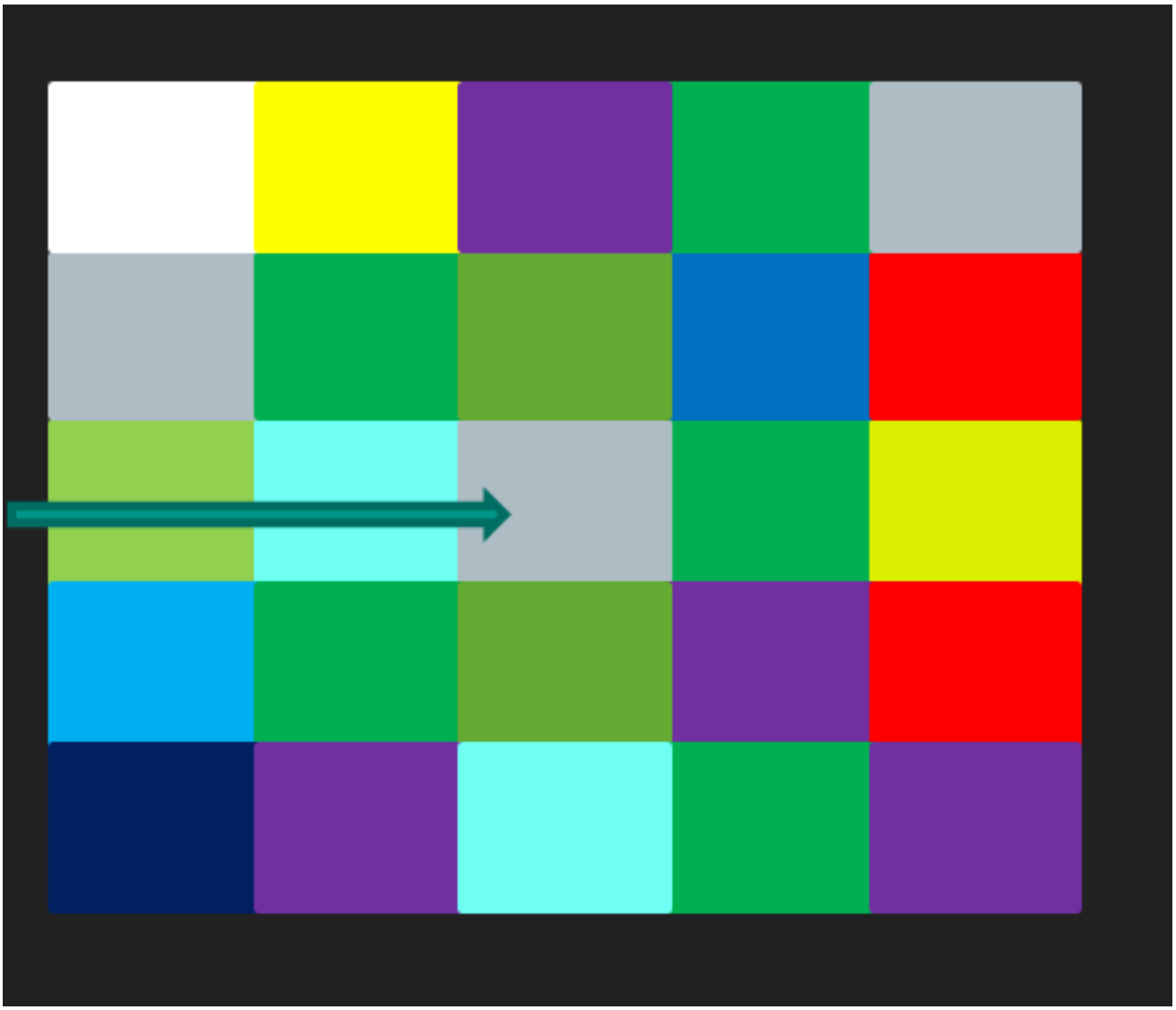


Рисунок 2.2.5.3. Насправді, для розмиття центрального фрагменту з радіусом 2 потрібно брати до уваги 25 фрагментів.

В даній дипломній роботі використовується Bilateral Gaussian blur.

Розмиття по Гаусу – це результат розмиття зображення, використовуючи функцію Гауса. Основна ідея в тому, щоб не просто брати середнє значення з декількох сусідніх фрагментів, а також взяти до уваги віддаленість кожного фрагмента від центрального. Чим ближче фрагмент до центрального, тим сильніше його вплив на колір центрального фрагмента.

Для оптимізації замість одного проходу використовується два: горизонтальний та вертикальний. Це дає змогу досягти такого ж результату, як і при одному проході, але набагато швидше, адже при використанні двох проходів потрібно набагато менше разів вичитувати значення з текстури, що може бути досить довго для текстур великої розмірності.

Використовуючи два проходи для горизонтального blur-у радіусом $R = 2$ потрібно розрахувати всього п’ять фрагментів: чотири сусідніх (по два с кожного боку, зліва та справа), та центральний (рисунок 2.2.5.1.). Після проходження етапу горизонтального blur-у для кожного із фрагментів наступає етап вертикального blur-у, тобто для вертикального blur-у радіусом $R = 2$ потрібно розрахувати також п’ять фрагментів: чотири сусідніх (по два з кожного боку, зверху та знизу), та центральний (рисунок 2.2.5.4). Завдяки такому підходу, замість зчитування та обчислення матриці 5x5, тобто 25 змінних для кожного фрагменту, буде використовуватись 10 змінних (5 для горизонтального проходу, 5 для вертикального). (рисунок 2.2.5.6)

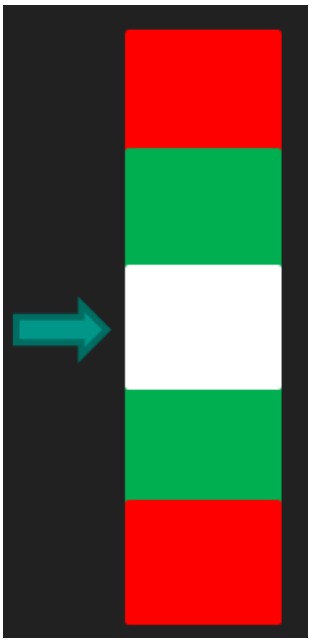


Рисунок 2.2.5.4. Вертикальний прохід для центрального фрагменту до розмиття

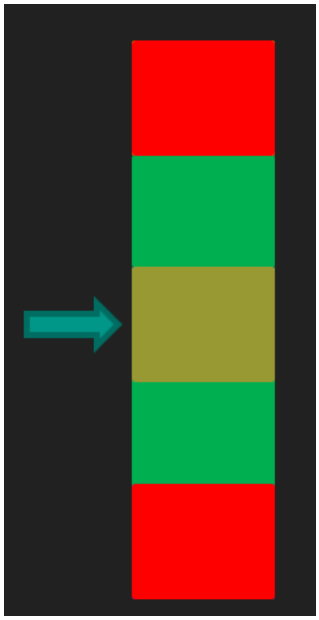


Рисунок 2.2.5.5. Вертикальний прохід для центрального фрагменту після розмиття

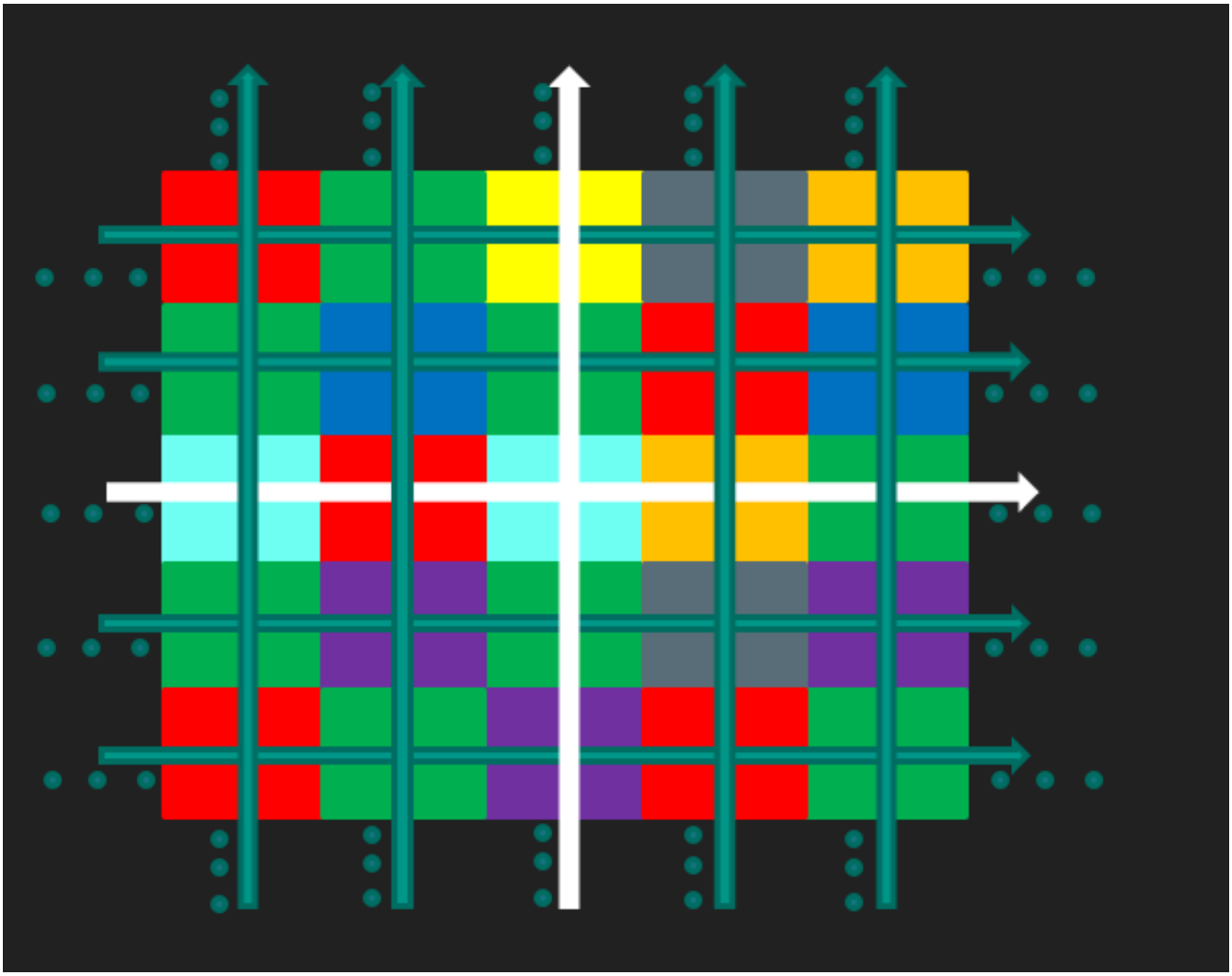


Рисунок 2.2.5.6. Спочатку виконується горизонтальній прохід для кожного фрагменту, потім вертикальний прохід для кожного фрагменту. Результат такий самий, як і при обробці 25 фрагментів, але обчислень менше.

Так як спочатку виконується горизонтальний blur, в текстурі зберігається не просто колір кожного фрагменту, а колір кожного фрагменту, вже «змішаний» по горизонталі з сусідніми. Отже наступний крок – вертикальний blur, що розмиває по вертикалі уже попередньо розмиті по горизонталі фрагменти. Завдяки цьому прийому кількість обчислень та зчитувань з текстури зменшується в декілька разів без втрати візуальної якості зображення, тому ця оптимізація є необхідною для real-time rendering, так як час обчислень відіграє важливу роль.

Bilateral blur означає, що до уваги потрібно також прийняти різницю фрагментів по глибині. Тобто, якщо фрагменти досить сильно відрізняються по глибині, це означає, що вони знаходяться далеко один від одного в 3D-просторі, отже не потрібно розмивати їх значення, так як це буде некоректно. Для цього потрібно додати одну невеличку перевірку по глибині для blur-у:

$$mod(Z_f - Z_c) < Diff \quad (2.9)$$

, де:

- $Z_f -$ значення глибини сусіднього фрагменту, зчитаної з текстури позицій
- $Z_c -$ значення глибини центрального фрагменту
- $Diff -$ максимально допустима різниця між глибинами.

Якщо глибина сусіднього фрагменту набагато більша за глибину центрального, то цей сусідній фрагмент не використовується для розмиття.

Рисунок 2.2.5.6 SSAO текстура після розмиття

Рисунок 2.2.5.7 SSAO текстура після розмиття

2.3 Затемнення сцени

В дипломній роботі використовується модель затінення Blinn-Phong, так що фінальний колір по моделі Blinn-Phong вираховується з наступної формули (вважаючи, що колір світла білий, тобто $C_l = (1, 1, 1)$, його можна не брати до уваги в фінальних розрахунках) :

$$L_{bp} = L_a + L_d + L_s \tag{2.10}$$

, де:

- L_{bh} - колір фрагменту по моделі Blinn-Phong
- L_a- ambient освітлення
- L_d- diffuse освітлення
- L_s- specular освітлення

$$L_a = C * A \tag{2.11}$$

, де:

- C – константний колір об’єкту
- $A -$ фактор для зменшення впливу ambient освітлення на об’єкт, фактично використовується значення 0.3

$$L_d = \omega_i * n * C \tag{2.12}$$

, де:

- $\omega_i -$ напрямок світла, що спрямований від поверхні до точкового джерела світла.
- $n -$ нормаль до поверхні
- C – константний колір об’єкту

$$L_s = (H * n)^B \quad (2.13)$$

, де :

- H –half-вектор (див. Розділ 1.2.3)
- $n -$ нормаль до поверхні
- $B -$ фактор, який показує наскільки багато світла дзеркально відображається від поверхні, фактично дорівнює 16

Колір, нормаль, та позиція фрагмента вже відомі, їх можна зчитати с відповідних текстур. Напрямок світла константний і також відомий, так як джерело світла не рухається. Позиція камери задається кожний кадр в залежності від того, де знаходиться камера в даний момент часу. Щоб отримати фінальну затемнену сцену потрібно прийняти до уваги SSAO текстуру, яку щойно було створено. Після розрахунку L_{bh} знаходимо фінальний колір фрагменту:

$$C_f = L_{bp} * AO^2 \quad (2.14)$$

, де

- $C_f -$ фінальний колір фрагменту
- $AO -$ Значення, зчитане с текстури SSAO для даного фрагменту. ($1 - occlusion -$ фактор)

В формулі використовується occlusion-фактор в квадраті для того, щоб ділянки, які мають бути трохи темнішими, стали більш темнішими, так що залежність стала квадратичною, адже це виглядає більш натурально.

Рисунок 2.3.1 Сцена, затемнена за допомогою Blinn-Phong shading model, без ambient occlusion

Рисунок 2.3.1 Сцена, затемнена за допомогою Blinn-Phong shading model, з ambient occlusion

Рисунок 2.3.2 Сцена, затемнена за допомогою Blinn-Phong shading model, без ambient occlusion

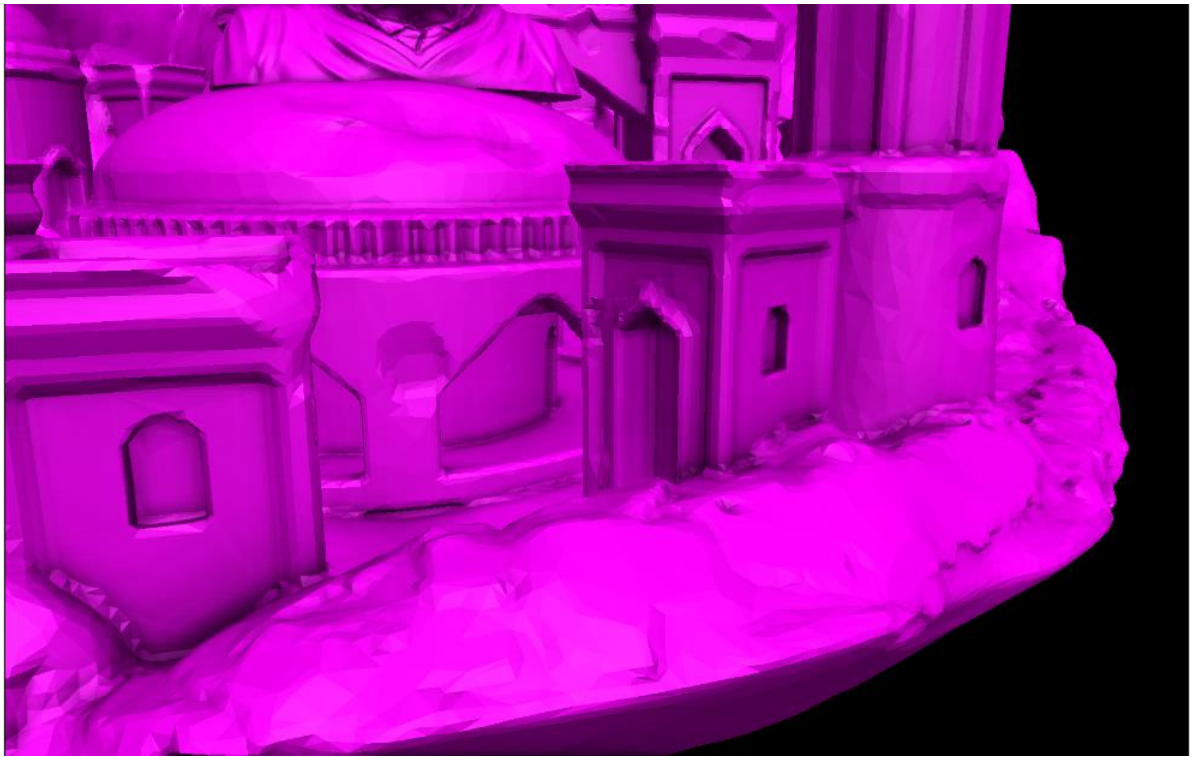


Рисунок 2.3.3 Сцена, затемнена за допомогою Blinn-Phong shading model, з ambient occlusion

Рисунок 2.3.3 Сцена, затемнена за допомогою Blinn-Phong shading model, без ambient occlusion

Рисунок 2.3.4 Сцена, затемнена за допомогою Blinn-Phong shading model, з ambient occlusion

ВИСНОВКИ

Дослідивши алгоритм моделі затінення SSAO (screen space ambient occlusion) можна стверджувати, що завдяки різного роду апроксимаціям можливо здобути правдоподібний результат затінення сцени за менший час. Так як для індустрії рендерингу в реальному часі швидкість виконання алгоритму є одним із найважливіших критеріїв, то чим швидше виконується алгоритм, тим краще, якщо він надає результати, близькі до тих, що обраховуються набагато довше.

Виявлено, що за допомогою методу Монте-Карло можна замінити «тяжке» інтегрування по півсфері на сукупність декількох точок (у дипломній роботі було використано 32 точки), які можна хаотично обертати для кожного фрагмента, таким чином симулюючи інтегрування по всій півсфері.

За допомогою моделі затінення Blinn-Phong можна отримати результати, які навіть краще описують поведінку світла (хоч і не фізично правильно), але обраховуються за менший проміжок часу.

Використовуючи deferred shading алгоритм дозволяє в декілька разів зменшити час, що потребується для розрахунку SSAO, так як усі необхідні дані вже будуть використовуватись для освітлення, а отже можуть бути використані знову для SSAO.

Завдяки використанню двох проходів для Bilateral Gaussian Blur економиться значна кількість часу, адже «повільна» для відеокарти процедура вичитки даних з текстури виконуватиметься в декілька разів менше.

Отже, SSAO – це техніка, алгоритм якої можна і потрібно змінювати в залежності від очікуваного результату та галузі. Алгоритм завжди можна

поліпшувати, додаючи більш точні обчислення за рахунок потужності апаратного забезпечення, або навпаки, додавати все більше апроксимацій та обчислювати все скоріше, але за рахунок візуальної якості.

Алгоритм, представлений в даній роботі, працює в 60 кадрів в секунду, використовуючи певну сцену та моделі затінення, та видає непоганий візуальний результат, що може влаштовувати деякі візуалізатори або ігри.

Також для більш фізично правильного зображення occlusion-фактор можна додавати тільки до ambient-компоненти моделі затінення, а не до всього результату. Якщо використовувати біль фізичну правильну модель затінення (наприклад PBR – Physically Based Rendering), алгоритм можна змінювати в залежності від того, чого не вистачає: візуальної коректності зображення або швидкості обчислень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Joey de Vries, стаття на тему «SSAO»: https://learnopengl.com/Advanced-Lighting/SSAO
2. Joey de Vries, стаття на тему «Normal Mapping»: https://learnopengl.com/Advanced-Lighting/Normal-Mapping
3. Joey de Vries, стаття на тему «Phong shading model»: https://learnopengl.com/Lighting/Basic-Lighting
4. John Chapman, стаття на тему «SSAO»: http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html

5. VRArviLab, стаття на тему «AMBIENT OCCLUSION: AN EXTENSIVE GUIDE ON ITS ALGORITHMS AND USE IN VR»: https://vr.arvilab.com/blog/ambient-occlusion
6. Jason Gregory, книга “Game Engine Architecture. Third Edition” ст. [360-380], [619-700]
7. OpenGL 4 Shading Language Cookbook ст.[111-120], ст.[146-150]
8. DirectX tutorial: http://www.rastertek.com/dx11tut50.html
9. Документація DirectX: https://docs.microsoft.com/en-us/windows/win32/direct3d11/
10. Форум на тему bilateral blur: https://stackoverflow.com/questions/6538310/anyone-know-where-i-can-find-a-glsl-implementation-of-a-bilateral-filter-blur/6538650
11. Форум на тему “two pass Gaussian blur”: https://stackoverflow.com/questions/33569396/correctly-implement-a-2-pass-gaussian-blur
12. Форум на тему “Gaussian blur in openGL”: https://stackoverflow.com/questions/44829671/smooth-blur-gaussian-in-opengl-es-2-0
13. Nvidia, книга GPU Gems 3, розділ 40 «Incremental Computation of the Gaussian» https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-40-incremental-computation-gaussian
14. Стаття Palisade на тему «метод Монте-Карло» : https://www.palisade.com/risk/monte_carlo_simulation.asp

ДОДАТОК

Презентація дипломної роботи на тему:

Імплементація моделей затінення ambient occlusion для візуалізації 3d-сцен

Рівняння рендеру

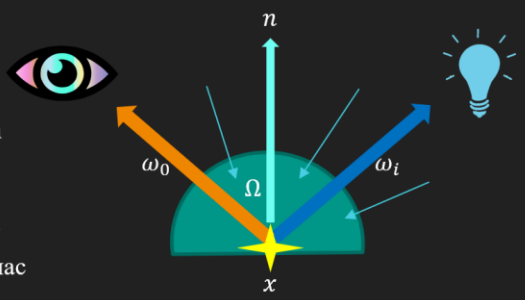
$$L_o(x, \omega_0, \lambda, t) = L_e(x, \omega_0, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_0, \lambda, t)\, L_i(x, \omega_i, \lambda, t)(\omega_i * n)\, d\omega_i$$

$L_o(x, \omega_0, \lambda, t)$ – це кількість вихідного випромінювання (spectral radiance) с довжиною хвилі λ, спрямованого вздовж напрямку ω_0 в час t з заданої точки x

$L_e(x, \omega_0, \lambda, t)$ – випромінюване світло

$f_r(x, \omega_i, \omega_0, \lambda, t)$ – bidirectional reflectance distribution function (BRDF - двонаправлена функція розподілу відбиття), кількість (пропорція) випромінювання, відбитого з напрямку ω_i у напрямку ω_0 в точці x, в час t на довжині хвилі λ

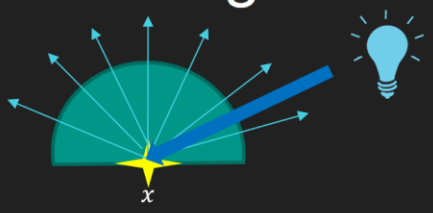
$L_i(x, \omega_i, \lambda, t)$ – кількість випромінювання з довжиною хвилі λ що надходить до точки x з напрямку ω_i.



$\omega_i * n$ – коефіцієнт ослаблення отриманого випромінювання по заданому куту між напрямками ω_i та n. Часто записується як $\cos \Theta_i$

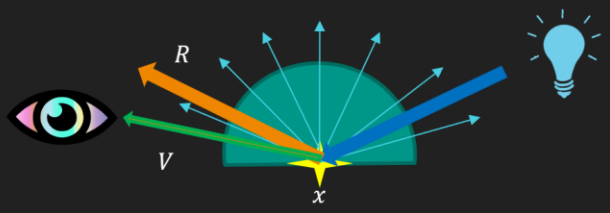
Апроксимація рівняння рендеру для real-time rendering

Lambertian reflectance

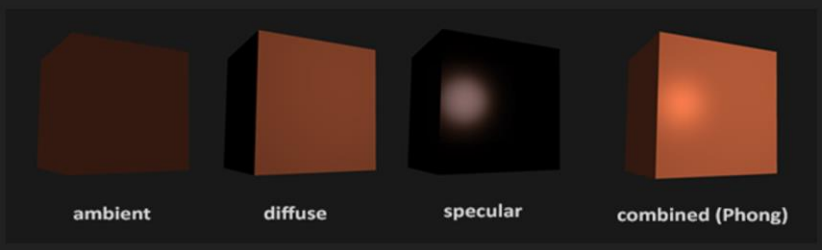


Phong shading model

Blinn-phong shading model



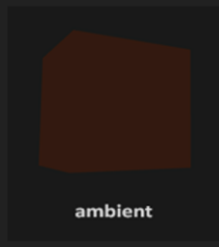
Phong shading model



Phong shading model

L_a- ambient компонента зазвичай дорівнює альбедо об'єкта, тобто його кольору, тобто $L_a = C * C_l * A$, де:

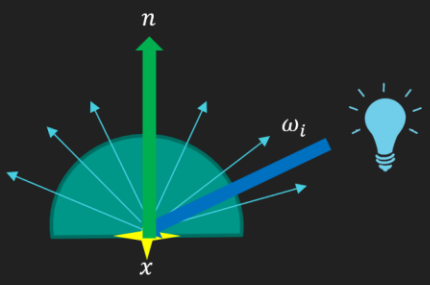
- C – константний колір об'єкту
- C_l – колір світла
- A – фактор для зменшення впливу ambient освітлення на об'єкт

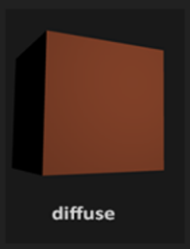


Phong shading model

$L_d = \omega_i * n * C * I_l$, де:

- ω_i – напрямок світла, що спрямований від поверхні до точкового джерела світла.
- n – нормаль до поверхності
- C – константний колір об'єкту
- I_l – інтенсивність світла

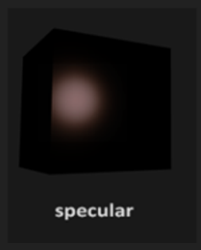


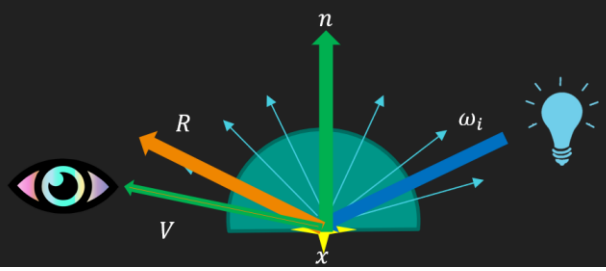


Phong shading model

$L_s = (V * R)^B * I_l * C_l$, де:

- V – вектор напрямку від точки освітлення до спостерігача (ока)
- R – вектор напрямку дзеркально відбитого промені світла в точці освітлення навколо нормалі до поверхності
- n – нормаль до поверхності
- C_l – колір світла
- I_l – інтенсивність світла
- B – фактор, який відображає наскільки багато світла дзеркально відображається від поверхні

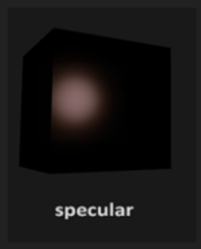


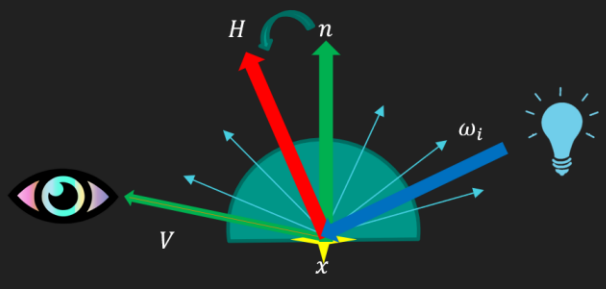


Blinn-phong shading model

$L_s = (H * n)^B * I_l * C_l$, де

- H –half-вектор
- n – нормаль до поверхності
- C_l – колір світла
- I_l – інтенсивність світла
- B – фактор, який показує наскільки багато світла дзеркально відображається від поверхні





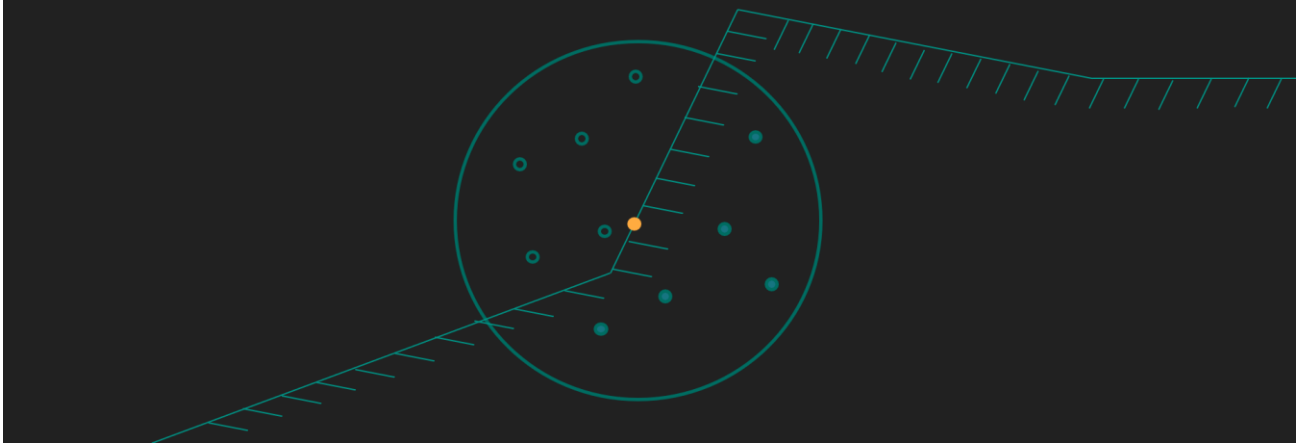
Ambient Occlusion

Ambient Occlusion

Ambient Occlusion

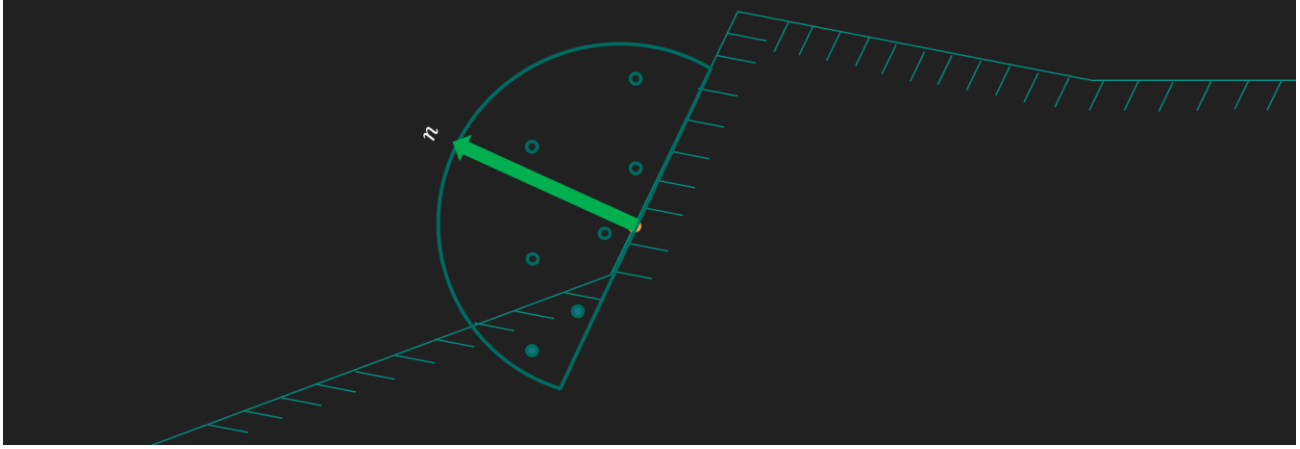
Ambient Occlusion

SSAO – Screen Space Ambient Occlusion



SSAO – Screen Space Ambient Occlusion

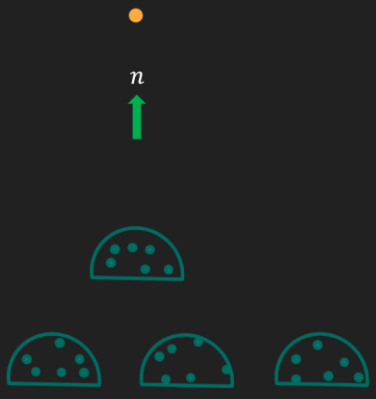
SSAO – Screen Space Ambient Occlusion



SSAO – Screen Space Ambient Occlusion

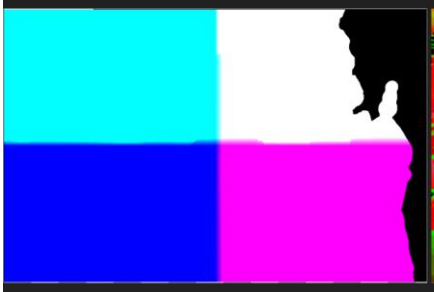
Для розрахунку SSAO потрібні наступні дані:

- Вектор позиції фрагменту в view-space (простір камери)
- Вектор нормалі до поверхності фрагменту в view-space
- Альбедо колір фрагменту
- Масив точок всередині півсфери (sample kernel)
- Вектор хаотичного повороту для того, щоб повернути sample kernel навколо нормалі



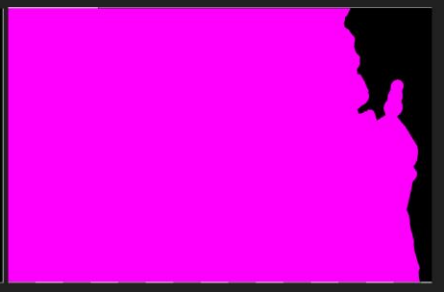
SSAO – Screen Space Ambient Occlusion

Текстура позицій: $P_v = M_v * M_w * P_m$



Текстура нормалей: $N_v = M_n * N_m$

Текстура альбедо

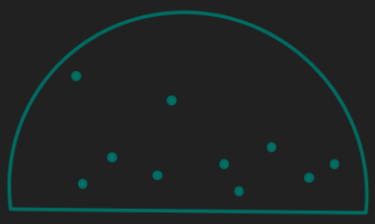


Sample kernel

$$P_s = normalize(P_r(random(-1,1), random(-1,1), random(0,1))) * random(0,1) * S$$

x та y – компоненти точки P_s лежить в межах $[-1; 1]$
z – компонента точки P_s лежить в межах $[0; 1]$

S- масштабування, яке забезбечує більшу розподіленість точок біля центру півсфери

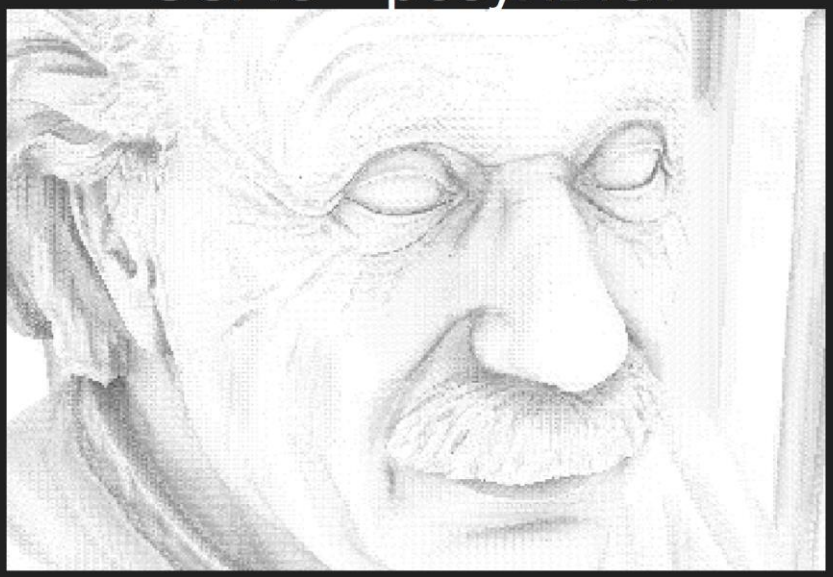


$$S = linear\_interpolation(0.1, 1, (\frac{i}{size})),$$ де

- i – номер точки, що наразі генерується
- $size$ –загальна кількість точок

SSAO - результат

SSAO - результат



Blur

Blur –техніка розмиття зображення.

Розмиття по Гаусу – це результат розмиття картинки, використовуючи функцію Гауса. Основна ідея в тому, щоб не просто брати середнє значення з декількох сусідніх фрагментів, а також прийняти до уваги віддаленість кожного фрагмента від центрального. Чим ближче фрагмент до центрального, тим сильніше його вплив на колір центрального фрагменту.

Bilateral blur означає, що до уваги потрібно також прийняти різницю фрагментів по глибині. Тобто, якщо фрагменти досить сильно відрізняються по глибині, це означає, що вони знаходяться далеко один від одного в 3D-просторі, тому не потрібно розмивати їх значення, тому що це буде некоректно.

Blur

Фінальне зображення до та після SSAO

Фінальне зображення до та після SSAO

Фінальне зображення до та після SSAO