Київський національний університет імені Тараса Шевченка
Механіко-математичний факультет
Кафедра алгебри та математичної логіки

Курсова робота
на тему:

3D реконструкція і склеювання даних з камери глибини

Студента четвертого курсу
Радченка Івана Сергійовича

Науковий керівник:
доктор фізико-математичних наук, доцент
Лавренюк Ярослав Васильович

КИЇВ — 2018

Зміст

1. Вступ 2
2. Загальний опис методу 3
3. Перетвореня карти глибини 4
4. Відслідковування камери 4
5. Об'ємна інтеграція 6
6. RayCasting 8
7. RayTracing 9
8. Висновок 14
9. Література 14

1. Вступ

Камера глибини (Kinect) використовує техніку шаблоного світла (structured light techique)[1], щоб генерувати в реальному часі карту глибини яка складається з відстаней від камери до точок фізичної сцени. Цю карту можна відобразити на множину відповідних точок у просторі (point cloud).

Хоча дані з Kinect є відносно якісними, проте вони мають багато шумів. Довжини зазвичай коливаються, а карти глибини містять багато "дірок", де не було отримано жодних результатів. Це можна побачити на Рис. 1 (B).

Щоб згенерувати геометрично точні 3D моделі для використання в таких сферах, як: іграх, фізиці, віртуальній реальності або доповненій реальності їх необхідно побудувати на отриманих зашумлених даних. Звичайно можна спробувати згенерувати полігональну стіку (mesh) на основі сильного припущення про неперервність сусідних точок в карті глибини. Однак, це призводить до зашумлених і низкьоякісних результатів, як показано Рис. 1 (C). Як видно це припущення створіює незакінчений mesh, на основі даних лише з однієї точки зору. Щоб створити закінчену або навіть "водонепронику" 3D модель, різні точки зору мають бути захопленні та склеїні в одну єдину конструкцію.

Рис. 1. Звичайна фотографія (A). Реконструкція нормалей (B) і поверхонь (C) з карти глибини за на основі припущення про непервність між сусідніми точками. 3D модель створена за допомогою KinectFusion показано нормалі поверхонь (D) і візуалізація з затемненям по Фонгу(Phong Shading) [12] (E)

В цій курсовій роботі розглянуто, без технічних подробиць реалізації, систему KinectFusion для швидкого створення детальних, геометрично точних 3D-реконструкцій з даних які реєструє камера глибини (Kinect) у реальному часі. Також представлений опис алогритму трасування променів та поєднання його з RayCasting за моєї реалізації, що є одним із етапів побудови 3D моделі.

2. Загальний опис методу

Система неперервно відслідковує 6 ступенів вільності(6DoF) Рис. 2 позиції камери і склеює нові точки сцени в єдину 3D модель. З кожною новою точкою зору реконструкція стає детальнішою. Дірки заповнюються, і модель стає більш завершеною.

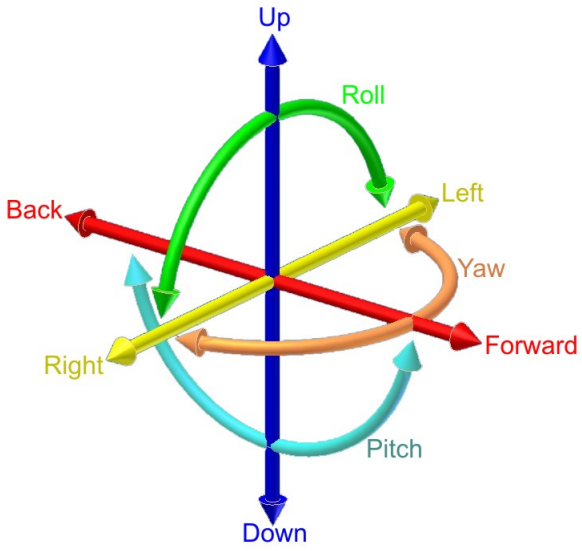

Рис. 2. 6 Degrees of freedom

Загалом система складається з 5 основних кроків:

(1) **Перетвореня карти глибини.** Перетворення карти глибини теперішнього кадру в 3D точки (вершини) та знаходження нормалей до поверхонь в заданих вершинах у системі координат відносно камери.
(2) **Відслідковування камери.** На стадії відслідковування, ізометричне 6DOF перетворення знаходиться для співставлення точок теперішнього та минулого кадру за допомогою ітеративного алгоритму найближчих точок(ICP)[2].
(3) **Об'ємна інтеграція.** Замість зливання точок з point cloud'ів або стоверння mesh'у, використовується об'ємне представлення[3]. Точки з теперішнього кадру перетворюються в точки у глобальній системі координат, і кожен воксель у 3D сітці оновлюється. Воксель зберігає у собі середнє значення дистанції до точок у середені нього.
(4) **Raycasting.** Нарешті, об'ємна сітка піддається raycasting'у для того, щоб вилучити данні про те, що саме видно користувачеві. Оскільки raycasting відбувається відносно теперішньої позиції камери в глобальних координатах, то дані в 3D сітці можна брати в якості штучної карти глибини, котра може бути використана як менш зашумлемлена і більш загальна для наступної ітерації

ICP. Це дозволяє виводити менш зашумлене зображеня створеного завдяки raycasting'у, на відміну від передавання даних з кадру до кадру.

(5) **Raytracing.** У цій частині алгоритму 3D модель розфарбовується згідно отриманої карти кольорів та геометричних властивостей. Це дозволяє побудувати якісне зображеня на основі оптичних властивостей об'єктів, таких як дифузне розсіювання світла, блиск, відзеркалення, заломлення тощо. Це стосується як реальних так і віртуальних об'єктів, якими можна доповнити реальність.

3. Перетвореня карти глибини

У час i, паралельно обробляємо кожен піксель $u = (x, y)$ у теперішній карті глибини $D_i(u)$. Для кожного пікселя знаходимо відповідну 3D координату у системі координат камери використовуючи калібруючу матрицю K, $v_i(u) = D_i(u)K^{-1}[u, 1]$. Результати збергіаємо у карту вершин V_i.

Відповідні вектори нормалей для кожної вершини вираховуються паралельно використовуючи сусідні вершини: $n_i(u) = (v_i(x+1, y) - v_i(x, y)) \times (v_i(x, y+1) - v_i(x, y))$, тоді нормалізуються $n_i/||n_i||$. Результати збергіаємо у карту нормалей N_i.

6DOF позиція камери в час i це ізометричне перетворення,яке можна задати матрицею $T_i = [R_i|t_i]$ яка містить 3×3 матрицю R_i повороту і 3 вимірний вектор зсуву t_i. Використовуючи це, вершини і нормалі можуть бути перетворенні у глобальну систему координат: $v_i^g(u) = T_i v_i(u)$, $n_i^g(u) = R_i n_i(u)$ відповідно.

4. Відслідковування камери

ICP це популярний і добре вивчений алгоритм для 3 вимірного вирівнювання фігур [4], див Рис. 3.

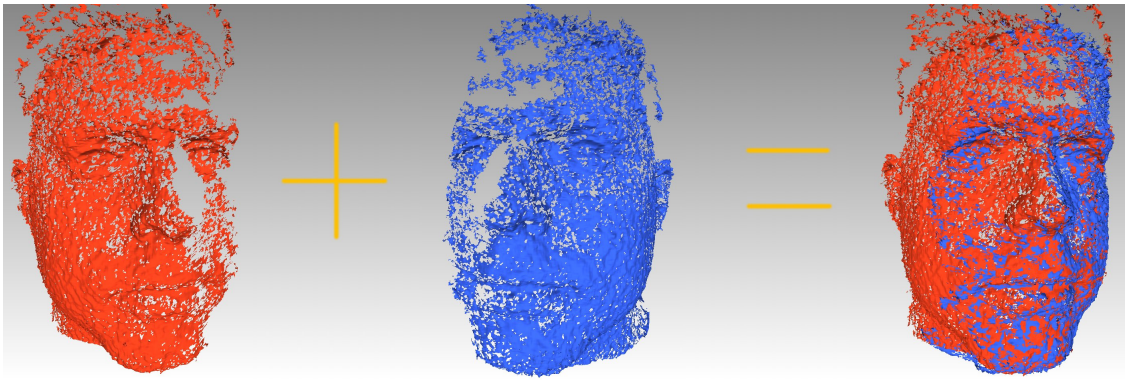

Рис. 3. приклад роботи ICP

У KinectFusion ICP використовується для відслідковування позиції камери у кожному новому кадрі. ICP повертає одне 6DOF перетворення, котре вирівнює точки

теперішнього кадру близько до точок попереднього кадру. Це перетворення поєднується з попередніми даючи позицію камери T_i.

Важлим першим кроком ICP є знаходження точок відповідності між точками у час i з точками у час $i-1$. У нашій системі ми використовуємо projective data association [5], щоб знайти ці відповідності. Псевдо код показаний у Код 1. На основі минулої позиції камери T_{i-1}, паралельно кожну точку v_{i-1} перетворюємо у точку в системі координат камери, тоді перспективно проєктуємо отриману точку у координати зображення з камери. Використовуємо цю двовимірну точку, щоб знайти вершину і нормаль у V_i та N_i відповідно. Тоді для кожної точки перевіряємо їх порівнюваність з відповідними точками, щоб виключити зайві, для початку перетворюємо обидві точки у глобальні координати, тоді порівнюємо відстань між ними і кут з якоюсь заданою точністю. Необхідно зауважити, що T_i спочатку прирівнюється до T_{i-1} і оновлюється з кожною ітерацією ICP.

Код 1 Projective point-plane data association. Тут ϵ і δ задані точності, для відстаней і кутів відповідно.

Для кожного піксель $u \in$ з карти глибини D_i **паралельно виконувати**:
  Якщо $D_i(u) > 0$:
    $v_{i-1} \leftarrow T_{i-1}^{-1} v_{i-1}^g$
    $p \leftarrow$ перспективно проєктуємо вершину v_{i-1}
    Якщо $p \in V_i$:
      $v \leftarrow T_i V_i(p)$
      $n \leftarrow R_i N_i(p)$
      Якщо $||v - v_{i-1}^g|| < \epsilon$ **і** $abs(n * n_{i-1}^g) < \delta$:
        точки відповідності знайдені

Маючи цю множину точок відповідності, ICP з кожною ітерацією повертає одну матрицю перетворення T, яка мінімізує point-to-plane помилку у метриці [6], визначену як суму квадратів дистанції між кожною точкую в теперішньому кадрі і дотичною площиною відповідної точки у попередньому кадрі:

$$argmin \sum_{D_i(u)>0} ||(Tv_i(u) - v_{i-1}^g(u)) * n_{i-1}^g(u)||^2$$

5. Об'ємна інтеграція

Після того як ми знайшли глобальну позицію камери використовуючи ICP, кожна координата з картинки, що повертає камера може бути переведене у глобальний координатний простір. Тоді інтегруємо ці дані у єдину об'ємну сітку на основі [7]. 3D куб фіксованого розміру є наперед заданим, і відповідає розмірам фізичного сцени. Цей куб ділиться на 3D сітку вокселей. Вершини з глобальних координат інтегруються у вокселі використовуючи варіант Знакової Функції Дистанції (SDFs) [8], що визначає відносну відстань до поверхні. Ці значення є позитивні попереду поверхні, і негатвні ззаду, і нульовою на ній де значення змінюють знак.

На практиці, необхідно використовувати Обрізану Знакову Функцію Відстані (TSDFs) [9]. Це представлення корисне для камери глибини, неявно кодує дані відстаней, ефективно заповнює отвори як тільки додані нові вимірювання і неявно зберігає геометрію поверхні.

Алгоритм оновлює значення TSDF і дозволяє неперервно дискретизувати дані про поверхню у сітку вокселів на основі карт глибин.

Псевдокод у Код 2. показує основні кроки алгоритму. Паралельно проходимо по всіх шматочках на осі Z, знаючи розмірність об'єму реконструкції і фізичні виміри, кожне дискретне розташування в 3D сітці може бути переведене у глобальні координати. Дистанція від камери до вершини (x, y) може бути вирахувана. Цю 3D вершину можна також перспективно спроєктувати назад на картинку, щоб подивитись теперішню відстань з камери глибини. Цю різню між довжинами вираховуються і кладуть у нове SDF значеня для вокселя див Рис. 4. Тоді ця величина нормується і усереднюється згідно попередніх значень, використовуючи звичайне середнє зважене. Нові значеня ваги і усереднене значеня TSDF кладуться у воксель.

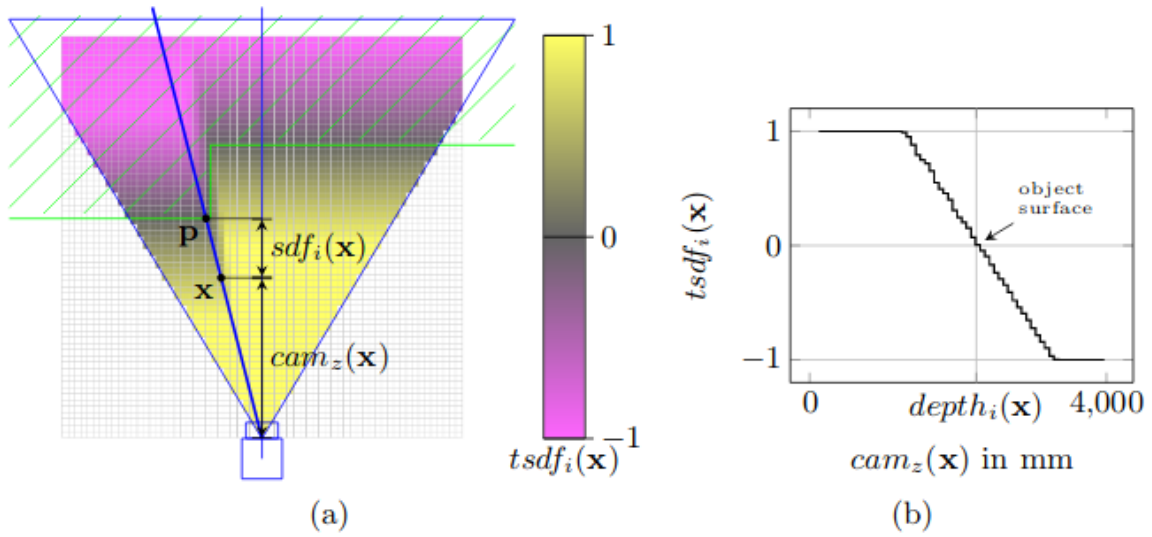

Рис. 4. 2D TSDF приклад (a) Цільний об'єкт (зелений), камера з деяким полем зору (синій), і вид зверху 3D сітки. Знакова відстань вокселя x визначена як глибина точки поверхні p і відстань до вокселя $cam_z(x) = ||t_i - v^g||$. (b) Графік TSDF вздовж променя з камери через p з $max\_truncation = 1000mm$. Поверхня об'єкта у нулі.

Код 2 TSDF інтеграція.

Для кожного вокселня g у об'ємному шматочку x, y **паралельно виконувати**:
  Поки проходимо від передньої частини шматочку до задньої **виконувати**:
    $v^g \leftarrow$ перетворюємо g з сітки у глобальні координати.
    $v \leftarrow T_i^-1 v^g$
    $p \leftarrow$ перспективно конвертуємо v
    Якщо v у полі зору камери:
      $sdf_i \leftarrow ||t_i - v^g|| - D_i(p)$
      Якщо $sdf_i > 0$:
        $tsdf_i \leftarrow min(1, sdf_i / max\_truncation)$
      Інакше:
        $tsdf_i \leftarrow max(-1, sdf_i / min\_truncation)$
      $w_i \leftarrow min(max\_weight, w_{i-1} + 1)$
      $tsdf^{avg} \leftarrow \frac{tsdf_{i-1} w_{i-1} + tsdf_i w_i}{w_{i-1} + w_i}$
      зберігаємо w_i і $tsdf^{avg}$ у воксель g

6. RayCasting

Псевдо код RayCasting'у представлений у Код 3. Паралельно для кожного пікселя зображення визначаємо його колір. Знаючи початкову позицію і напрям променю, проходимо по всіх вокселях, що лежать на промені, і беремо звідти позицію поверхні на нульовом перетині(тобто значення де TSDF змінює знак). Остаточний перетин променю з поверхнею вираховується використовуючи просту лінійну інтерполяцію на трьох точках з кожної сторони і нульового перетину. Припускаючи, що градієнт перпендикулярний до дотичної поверхні, нормаль вираховується як похідна TSDF у нульовому перетині [10]. Після того, як перетин знайдений, необхідно знайти вершину і нормаль, які разом з напрямком променя можна використати як параметри для вираховування світла на вихідному пікселі, за допомогою трасування променів.

Код 3 RayCasting.

1: **Для** кожного пікселя u у вихідному зображенні **паралельно виконувати**:
2: **Якщо** $max\_len == NULL$
3: $max\_len \leftarrow inf$
4: **Якщо** $ray^{dir} == NULL$ і $ray^{len} == NULL$:
5: $ray^{start} \leftarrow [u, 0]$ конвертується у позицію в сітці.
6: $ray^{next} \leftarrow [u, 1]$ конверутється у позицію в сітці.
7: $ray^{dir} \leftarrow (ray^{next} - ray^{start})$
8: $ray^{len} \leftarrow 0$
9: $g \leftarrow$ перший воксель вздовж ray^{dir}
10: $m \leftarrow$ перетворюємо глобальні координати мешу у позицію в сітці
11: $m^{dist} \leftarrow ||ray^{start} - m||$
12: **Поки** воксель g у об'ємі **виконувати**:
13: $ray^{len} \leftarrow ray^{len} + 1$
14: $g^{prev} \leftarrow g$
15: $g \leftarrow$ переходимо до наступного вокселя на шляху ray^{dir}
16: **Якщо** нуль вокселя лежить від g до g^{prev}:
17: $p \leftarrow$ вилучаємо трилінійно інтерпольовану позицію в сітці
18: $v \leftarrow$ перетворюємо p з позиції в сітці у глобальну 3D позицію
19: $n \leftarrow$ вилучаємо градієнт поверхні як $\nabla tsdf(p)$
20: **Повертаємо** v, n, ray^{dir}
21: **Якщо** $ray^{len} > m^{dist}$ або $ray^{len} > max\_len$:
22: **Повертаємо** v, n, ray^{dir}

Дистанції від вершини мешу до камери вираховується у координатах сітки (Код 3 лінії 7 і 8). Ця дистація вираховується як додаткова умова зупинки променя (Код 3

ліня 19), дозволяє вирахувати невідповідності між об'ємним і mesh представленням поверхонь.

7. RayTracing

Загальне, дифузне і блискуче світіння може бути вирахувано згідно реконструкції і віртуальної геометрії. Більш розвинене затінення може бути вирухувано прослідковуючи шлях вторинного відбиття для кожного променння. Тіні вираховуються після першого зіткнення променння з вокселем або mesh поверхнею (Код 3 лінія 13 і 19), слідкуючи за вториним променем від поверхні до позиції світла (використовуючи координати сітки). Якщо поверхня зтикається з променем до того, як він обірвиться, то вершина затінюється. Для відзеркалення , як тільки промінь зтикається з поверхнею, новий напрямок променння вираховуються на основі нормалі поверхні і напрямку променя.

Алгоритм трасування променів достатньо поділити на дві частини, вираховування світла та трасування променя, але знаходження відповідного напрямку променя вже написано у RayCasting, що дозволяє більш "дешевше" знаходити перетин променя з об'єктами сцени. Тому під "трасуванням променя", тут буде розумітися саме система яка поєднує вираховування світла та RayCasting для заданих v n ray^{dir}.

Карту світла $light\_map$ у Код 4 будемо вважати відомою і такою, що складається з множини структур $light$, які містять поля інтенсивність, тип світла загальне-ambient, точкове-point, напрямне-direction, його позицію, якщо це точкове світло, чи напрямок, якщо це напрямне світло. Такий поділ зумовлюється підходом моделі Фонга до освітлення.Аналогічно з $specular\_map$ вважаємо відомою і такою, що для заданої вершини v повертає $specular\_map(v) = s \in [-1, \infty]$ -1 - не блискуча, для $s > 0$, чим більше s тим блискучіша поверхня.

Для заданої точки і нормалі створіюємо загальну міру інтенсивності i від кожного джерела освітлення. Проходимо по $light\_map$ якщо тип світла $ambient$, то ми просто додаєм його до i це світло симулює "дешеве" відбиття світла від інших предметів.

Якщо світло точкове, то нам необхідно дізнатися його напрямок L за простою формулою $L = light.position - v$. Якщо світло напрямне, то його напрямок вже є відомим, тому $L = light.direction$. Тоді вираховуємо, чи потрапляє на дану точку відповідне джерело світла для цього знову запускаємо RayCasting, але для заданого напрямку L і мінімальної довжини променя 0.001, бо точка не може відкидати сам на себе тінь. Також необхідно підкреслити, що максимальна довжина променя теж задається $max\_len = 1$ для точкового світла і $max\_len = \infty$ для напрямного, це зумовлено тим, що напрямне світло немає кінця, а для точково, тим що за джерелом світла немає сенсу перевіряти. Якщо знаходиться якась вершина на шляху променя світла, то вплив даного джерела не враховується.

Далі нам необхідно вирахувати дифузне освітлення, для цього визначаємо скалярний добуток нормалі n і напрямку світла L, для того щоб знайти кількість світла яку вносить дане джерело, як $\frac{light.intensity*n\_dot\_l}{len(n)*len(L)}$, бо $cos(\alpha) = \frac{n\_dot\_l}{len(n)*len(L)}$ де α кут між нормалю і напрямком світла. Далі потрібно врахувати те, що ми відкидаємо від'ємні значеня $cos(\alpha)$, бо інакше світло досягає задньої частини поверхні.

Тепер необхідно врахувати блискучість об'єкту $s = specular\_map(v)$, для цього перевіряємо чи є він блискучим, якщо так, то знаходимо відбитий промінь $R = 2*n*(n, L) - L$, тоді знаходимо скалярний добуток відбитого променя на промінь, випущений з Kinect. Якщо цей добуток більший за нуль, бо ми знову не роздивляємося блискучість з оберненого боку поверхні, то інтнсивність блискучості вираховується як $i \leftarrow light.intensity*(\frac{r\_dot\_dir}{len(R)*len(ray^{dir})})^s$. Ця модель блискучості пояснюється тим, що чим дужче напрямок променя, що відбивається від поверхні співпадає з напрямком відбитого світла, якби поверхня була гладкою, тим блискучиший об'єкт. Для такого розуміння міри блиску підходить $cos^s(\alpha) = (\frac{r\_dot\_dir}{len(R)*len(ray^{dir})})^s$, α - це кут між R і ray^{dir}. Як видно з Рис. 5 чим блискучіший об'єкт, тим дужче він передає природу світла.

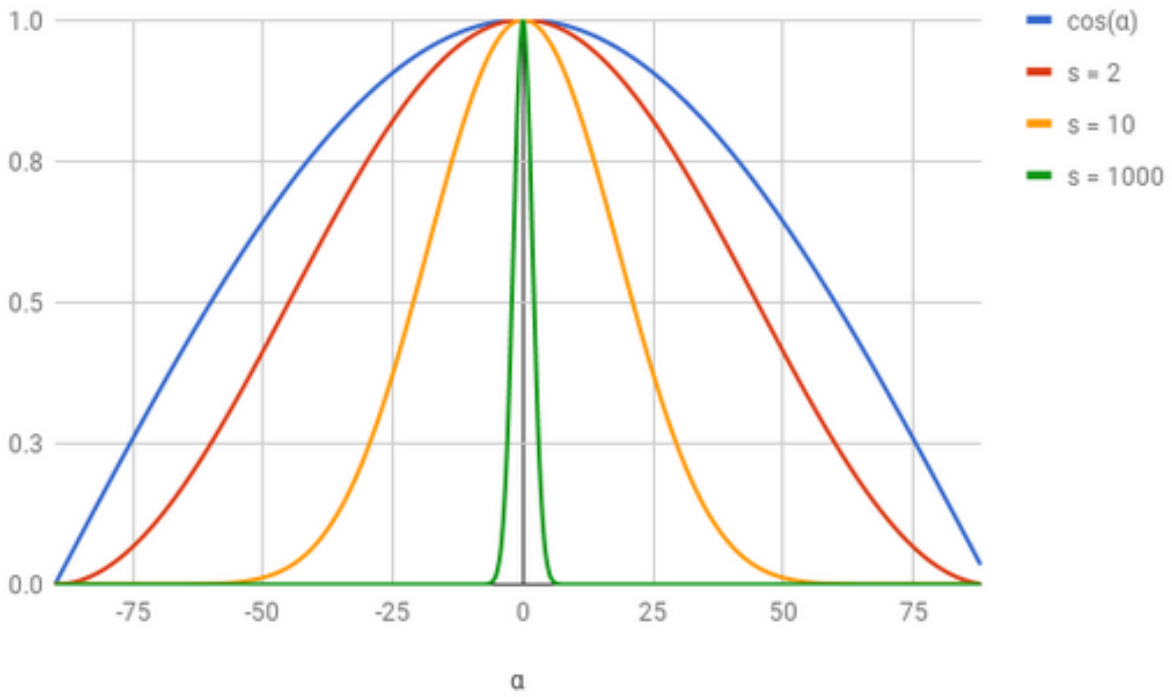

Рис. 5. графік $cos^s(\alpha)$

Нарешті інтенсивність освітлення i повертається для вирахування кольору у $RayTrace$. Псевдокод представлений у Код 4.

Код 4 ComputeLight.

ComputeLight для вершини, нормалі і напрямку променя v, n, ray^{dir}:
    $i \leftarrow 0$
    Для кожного світла *light* у карті світла *light_map*
        Якщо $light.type == ambient$:
            $i \leftarrow i + light.intensity$
        Інакше:
            Якщо $light.type == point$
                $L \leftarrow light.position - v$
                $max\_len = 1$
            Інакше:
                $L \leftarrow light.direction$
                $max\_len = infty$
            $min\_len \leftarrow 0.001$
            $shadow\_v \leftarrow raycasting(v, L, min\_len, max\_len)$
            Якщо $shadow\_v! = NULL$:
                перейти до наступного елементу циклу
            $n\_dot\_l \leftarrow (n, L)$
            Якщо $n\_dot\_l > 0$:
                $i \leftarrow i + \frac{light.intensity * n\_dot\_l}{len(n) * len(L)}$
            $s \leftarrow specular\_map(v)$
            Якщо $s! = -1$:
                $R \leftarrow 2 * n * (n, L) - L$
                $r\_dot\_dir = (R, ray^{dir})$
                Якщо $r\_dot\_dir > 0$:
                    $i \leftarrow light.intensity * (\frac{r\_dot\_dir}{len(R) * len(ray^{dir})})^s$
    Повернути i

Задача RayTracing полягає у тому, щоб визначити який саме колір необхідно повернути для заданої вершини. Тому якщо у RayTracing не надішло жодної вершини повертається фоновий колір. Інакше необхідно просто вирахувати якої саме інтенисивності є власний колір об'єкту $local\_color$, тому користуючись функцією *ComputeLight* описаною у Код 4, кожен канал кольору (RGB) множиться на інтенсивність освітлення.

Також потрібно враховувати зеркальність.$reflection\_map$ у Код 5 будемо вважати відомою і такою, що для будь-якої вершини v, $reflection\_map$ повертає значення

$reflection\_map(v) = r \in [0,1]$ $r = 0$ - повністю не зеркальна точка, $r = 1$ - зовсім зеркальна. Для того, щоб симулювати відзеркалення, потрібно власний колір $local\_color$ помножити по кожному каналу на значення незеркальності $1-r$ і додати колір відбиття $refl\_color$ помножений на значення зеркальності r.

Для того, щоб порахувати $refl\_color$, потрібно знайти відбитий промінь R від поверхні, та прослідкувати цей промінь до наступної вершини за допомогою $RayCasting$, тоді рекурсивно запускаємо $RayTracing$ для нової вершини та нового напрямку зменшуючи глибину рекурсії $depth$ на один, тим самим попереджуємо можливість потрапити у нескінчену петлю. Аналогічно можна побудувати прозорість об'єктів знаючи коефіцієнти заломлення поверхонь.

Код 5 RayTracing.

TraceRay для вершини, нормалі, трасуючого променю та глибини рекурсії $v, n, ray^{dir}, depth$:
    Якщо $v == NULL$
        Повернути Фоновий колір
    $local\_color \leftarrow color\_map(v) * ComputeLight(v, n, -ray^{dir})$
    $r \leftarrow reflection\_map(v)$
    Якщо $depth <= 0$ або $r <= 0$:
        Повернути $local\_color$
    $R \leftarrow 2 * n * (n, -ray^{dir}) + ray^{dir}$
    $min\_len \leftarrow 0.001$
    $max\_len \leftarrow \infty$
    $u, m, new\_R \leftarrow RayCasting(v, R, min\_len, max\_len)$
    $refl\_color \leftarrow TraceRay(u, m, new\_R, depth - 1)$
    Повернути $local\_color * (1 - r) + refl\_color * r$

На Рис. 6 показана блок схема відображення, яка дозволяє на основі звичайного mesh'у бути поєднаній з зображженям на основі raycasting'у, включаючи відображення віртуальних і реальних сцен з правильними фізичними властивостями. На першому кроці a), сцена на основі mesh'у виводться згідно глобальної позиції камери (T_i) і калібровці (K). Замість того, щоб вивести у фреймбуфер, вершиний буфер, нормалі до поверхонь і незатінені кольори кладуть у відповідні карти b), і використовують як вхідні дані для raycasting e).

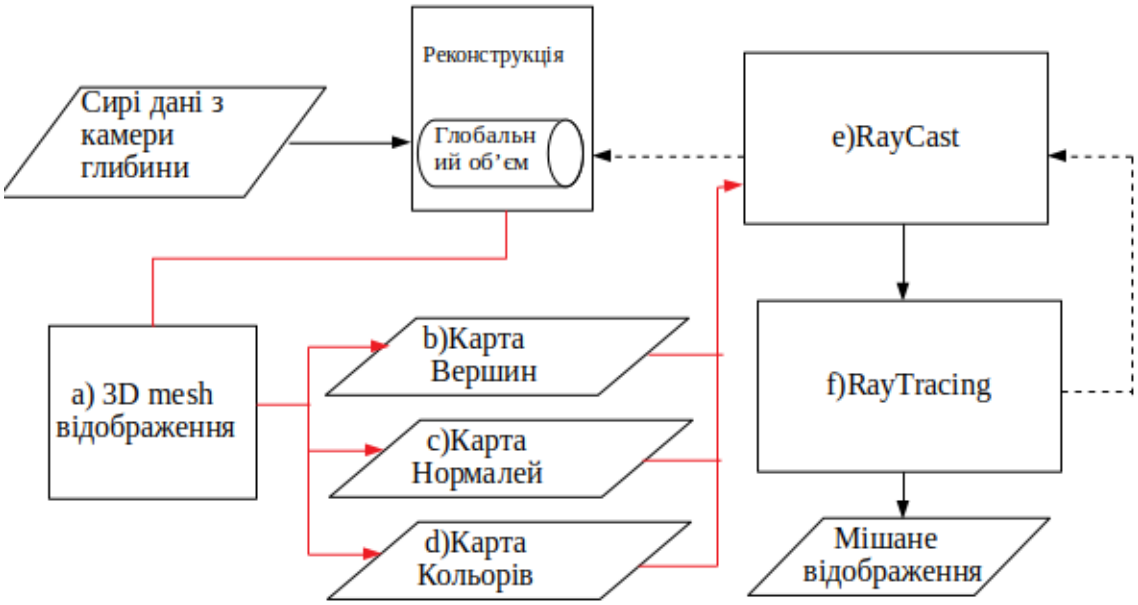

Рис. 6. Блок схема відображення

Це все дозволяє, поєднувати полігональну геометрію з правильним вираховуваними окклюзіями, і отримувати затінювання високої якості за допомогою трасуваня променів f).

Будь-який рух камери у 6 ступінях вільності може бути використаний для raycast'інгу об'єму включаючи вид від 3 обличчя, що дозволяє користувачу переміщатися в 3D моделі.

Інший плюс raycaster, це те що ми створюємо дані вискої якості для ICP відслідковування камери. Знайдені за допомогую raycast вершини і нормалі прирівнюються до глибин і карт нормалі з тієї ж позиції але з меншою кількості зашумлення і дірок ніж сирі дані з Kinect. Як показано [11], це дозволяє вирішиту проблему дрифту і знизити помилки ICP, відслідковуючи згідно моделі до якої вже застосувався raycast замість покадрового ICP відслідковування.

8. Висновок

У даній курсовій роботі було представлено алгоритм, який дозволяє робити 3D реконструкці приміщення. Було описано основні кроки, і більш детально показано кожен з них достатньо для абстрактного розуміння і наведено необхідну літературу для більш широкого ознайомлення з матеріалом. Також мною було побудовано алгоритм RayTracing та його зв'язок з RayCasting, що є останнім пунктом системи KinectFusion.

Очевидно, що результати роботи KinectFusion можна використати у віртуальній реальності, але більш цікавим є те, що цей алгоритм є простим для розширення реконструкції віртуальними елементами, що може бути використано у доповненій реальності з урахуванням геометрії реального простору.

Є багато реалізацій даного алгоритму, тому звичайному користувачеві достатньо мати ПК і камеру для 3D реконструкцій будь-чого, що знаходиться у приміщенні використовуючи лище переміщення камери.

9. Література

(1) B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns. Patent Appli- cation, 10 2008. WO 2008/120217 A2.

(2) P. J. Besl and N. D. McKay. A method for registration of 3D shapes.IEEE Trans. Pattern Anal. Mach. Intell.,14:239–256, February 1992.

(3) .B. Curless and M. Levoy. A volumetric method for building complex models from range images. ACM Trans. Graph. , 1996.

(4) R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In Proc. of the Int. Conf. on Computer Vision (ICCV) , 2011

(5) S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. 3D Digital Imaging and Modeling, Int. Conf. on , 0:145, 2001.

(6) Y. Chen and G. Medioni. Object modeling by registra- tion of multiple range images. Image and Vision Com- puting (IVC) , 10(3):145–155, 1992.

(7) B. Curless and M. Levoy. A volumetric method for building complex models from range images. ACM Trans. Graph. , 1996.

(8) S. Osher and R. Fedkiw. Level set methods and dynamic implicit surfaces . Springer, 2002.

(9) B. Curless and M. Levoy. A volumetric method for building complex models from range images. ACM Trans. Graph. , 1996.

(10) S. Osher and R. Fedkiw. Level set methods and dynamic implicit surfaces . Springer, 2002.

(11) R. A. Newcombe et al. KinectFusion: Real-time dense surface mapping and tracking. In ISMAR , 2011.
(12) Bui Tuong Phong, "Illumination for Computer Generated Pictures,"Comm. ACM, Vol 18(6):311-317, June 1975.