**Контрольна робота**

**з курсу "Ком'ютерна графіка"**

**студента 3 курсу**

**спеціальності «Інженерія програмного забезпечення»**

**Зимовця Руслана**

**1) Особливості зафарбування об'єктів. Який алгоритм краще застосувати? Чому?**

Існують наступні алгоритми зафарбовування об'єктів:

- Для зафарбовування ліній (якщо ми вважаємо це зафарбовуванням):

*Digital Differential Analyzer (DDA) algorithm*:

Маємо 2 точки. Визначаємо, що більше dx чи dy. Відповідно більшу величину використовуємо в якості кроку. Проходимось від x0 до x1 (або y0 до y1), при цьому на кожному кроці рахуємо прирости координат:

Xincrement = dx / (float) steps;

Yincrement = dy / (float) steps;

І зафарбовуємо відповідний піксель: putpixel(Round(x + Xincrement), Round(y + Yincrement));

**Недоліком є точність:** те, що ми використовуємо дійсні обчислення для роботи з цілочисельними значеннями.

На вході: 2 точки. . Далі . На наступному кроці обчислюємо прирости координат: . Проходимось в циклі і зафарбовуємо відповідні пікселі:

*for(int v=0; v < крок; v++)*

*{ x = x + ; y = y + ;*

*putpixel(Round(x), Round(y));*

*}*

До переваг віднесемо простоту алгоритму.

До недоліків даного алгоритму можна віднести його точність, оскільки ми використовуємо числа з плаваючою точкою для роботи з цілочисельними значеннями. Також ефективність алгоритму є відносно низькою. До того ж, можуть виникнути проблеми при відмалюванні вертикальних ліній. Товщина ліній обмежена 1 пікселем.

*Алгоритм Брезенхейма:*

Це алгоритм, що працює на цілих числах, та застосовується для побудування відрізку. При побудові зображення відрізка завжди вибирається найближчий по вертикалі піксель. Також, є алгоритм Брезенхема для побудови кіл, у якому основна увага віддається побудові одної четверті кола. А інші четверті будуються виходячи з симетрії кола.

До недоліків даного алгоритму можна віднести відсутність зглажування.

*Алгоритм Ву:*

Відрізняється від Алгоритму Брезенхейма тим, щовиконує зглажування, проте є повільнішим. На кожному кроці розглядаються 2 пікселі (замість одного) і підраховується похибка для коректного відображення природньої лінії. Між 2 пікселями, залежно від похибки, розподіляється інтенсивність зафарбування.

До переваг можна віднести присутність зглажування  
До недоліків – малу швидкість роботи (порівняно з *Брезенхеймом*)

Звісно вибір алгоритму залежить від області, де його буде застосовано. Частіше за все, зглажування є дуже важливим, тому часто найкраще використовувати алгоритм Ву.

- Для зафарбовування полігонів:

*Алгоритм сканування лінії (Scan Line Algorithm ):*

Це алгоритм замальовування полігонів шляхом горизонтальних ліній-сканерів. Обирається найбільша та найменша за у-координатою точки з множини вершин полігона, а також множина вершин сортується за у-координатою. Далі проходимось по всім парам точок , починаючи від максимальної за ординатою та до мінімальної і заповнюємо послідовність пікселів між цими парами точок.

*Алгоритм заповнення границі (Boundary Fill Algorithm)*

Це рекурсивний алгоритм для зафарбовування довільних полігонів. Починаємо з пікселя всередині полігона та заповнюємо внутрішню частину, рекурсивно переходячи на сусідні (в околі) точки. Вирішуємо, чи зафарбовувати піксель і продовжувати, спираючись на колір цього пікселя. Тип околу залежить від реалізації. Якщо використовуємо 4-зв'язну техніку, то можуть бути випадки, коли алгоритм не заповнить деякої області:

Background pattern

Description automatically generated with medium confidence

При 8-зв'язній техніці, такої проблеми не буде.

До недоліків можна віднести:

Ефективна 4-зв'язна техніка призводить до неповноти зафарбувань.

Здатність зафарбовувати полігони, які зовні оточені пікселями іншого кольору.

*Алгоритм Заповнення лініями*

Алгоритм подібний до попереднього, проте на кожному кроці замість рекурсивного розходження в усі сторони (по формі околу), він розходиться в боки горизонтальними лініями (також рекурсивно). Це дозволяє знизити використання автоматичної пам'яті (для рекурсивних викликів), чим досить добре підвищує ефективність алгоритму.

Переваги: висока ефективність, немає проблеми з 4-зв'язністю, як у попереднього алгоритма.

Підсумовуючи, можу сказати, що найчастіше у комп'ютерній графіці застосовується Алгоритм Заповнення Лініями порівняно з Алгоритмом заповнення границі. Це зумовлено його швидкістю а також відсутністю неприємних недостатків Алгоритму заповнення границі.

**2) Коли необхідно знаходити нормаль? Як її обрахувати ?**

У комп'ютерній графіці *нормаль* – це вектор, який описує орієнтацію поверхні. Нормалі широко використовуються, особливо при обрахунку інтенсивності освітлення поверхонь, затіненні, виявленні колізій тощо. З точки зору математики, нормаль – це вектор, перепендикулярний до певної поверхні.

Аби обчислити нормаль, можна скористатися поняттям векторного добутку:

Diagram

Description automatically generated

Модуль векторного добутку двох векторів (довжина нормалі) геометрично дорівнює площі паралелограма натягнутого на ці два вектори.

**3) Системи координат - світові, екранні. Як і де використовують ?**

Світові координати – це система координат, яка безпосередньо описує розташування зображування 3-вимірних зображуваних об'єктів у 3-вимірному просторі. З тривимірності випливає, що у кожної точки є 3 координати. Ці координати вимірюються відносно певної зафіксованої точки – центру, яка має координати (0, 0, 0). Існує окремо система координат камери, центр якої, у світових координатах, має координати камери «у світі» (тривимірному зображуваному просторі).

Екранні координати – це система координат, яка «розміщує точки» на комп'ютерному екрані та задає їм координати. Одиницею вимірювання координат є піксель. На цю систему координат ми здійснюємо проектування зображуваних об'єктів, коли зображаємо їх на екрані. Варто відмітити, що третя екранна координата кожної точки позначає глибину даної точки (глибину розміщення об'єкта, що займає цю точку).

Використовуються обидві системи координат при зображенні якихось трьовимірних об'єктів на екрані комп'ютера. Існує два підходи:

1. Нерухомимою є система координат камери, а світова система координат – рухома.
2. Нерухомою є світова система координат, в той чам як система координат камери – рухома.

Перетворення зі світових координат у екранні відбувається наступним чином:



Спочатку афінно перетворюємо світові у видові координати. Видові координати – «погляд камери».

Видові проектуємо на площину.

Проекцію далі перетворюємо в екранні координати афінним перетворенням: масштабуванням та зсувом.

**4) Чому використовуються матриці 4-го порядку у комп'ютерній графіці ?**

Афінне перетворення – відображення простору в себе, при якому паралельні прямі залишаються паралельними, пересічні – пересічними, мимобіжні – мимобіжними.

Загальний вигляд такого перетворення:

невироджена матриця, а

Однорідні координати – розширення звичайних n-вимірних координат шляхом додавання одного виміру, який відповідає за переміщення.

В однорідних координатах можна записати афінне перетворення наступним чином:

A picture containing text, clock, gauge

Description automatically generated

Як бачимо, такий вигляд нам дозволяє робити довільні афінні перетворення лише одним множенням на матрицю.

Тому у комп'ютерній графіці такий підхід дозволяє робити афінні перетворення над 3-вимірними об'єктами всього одним множенням на 4-вимірну матрицю.

Таким чином ми можемо здійснювати масштабування, зсув та поворот одночасно.

**5) Які необхідно виконати перетворення при оберті об'єкта навколо вектору ?**

Нехай треба здійснити обертання навколо вектору на кут .

Diagram

Description automatically generated

Дане афінне перетворення можна записати наступним чином :

Text, letter

Description automatically generated

(якщо в умові малося на увазі, що вектор є прикріпленим до певної точки, то поворот ми б здійснювали за алгоритмом повороту навколо прямої, що проходить крізь цю точку та має цей направляючий вектор. Алгоритм описано на слайдах презентації Lek04KG на слайдах 9-11)

(у афінних чи векторних просторах вектори самі по собі не є прив'язаними до точок, тому я й описав даний алгоритм як просто множення на матрицю. Мови про точку, до якої прикріплено вектор не йдеться в умові.)

**6) Які перетворення потрібно виконати при масштабуванні об'єкта?**

Спочатку визначеємо коефіцієнти масштабування кожної з координат: .

Далі визначаємо точку, відносно якої нам треба здійснити масштабування. Нехай її координати: .

Далі здійснюємо наступне афінне перетворення:

Text

Description automatically generated

Якщо масштабування відбувається відносно центру координат, то можна спростити цей вираз (бо ):

Text

Description automatically generated with medium confidence

По суті, масштабуючи відносно точки, ми спочатку виконуємо зсув таким чином, щоб ця точка перейшла у початок координат, а потім виконуємо масштабування відносно центра координат.

Тому, якщо ми робимо масштабування об'єкту відносно його ж позиції, то це можна інтуїтивно розділити на :

зсув об'єкта так, щоб він був відцентрований (у початку координат)

масштабування відносно центра.

Хоча таке розділення не є ефективним з точки зору алгоритмів (два множення матриці замість одного), проте дозволяє краще зрозуміти механізм масштабування відносно точки.

**7) Особливості перспективної та аксонометричної проекції. Як і коли застосовують?**

*Аксонометрична проекція* — різновид паралельної проекції, за якої всі проектуючі промені розміщуються під прямим кутом до площини проектування. Аксонометричні проекції показують зображення об'єкта, якщо дивитися з косого напрямку, щоб виявити всі три напрямки (осі) простору в одному малюнку.

*Особливості аксонометричної проекції:*

* Проекції відрізків паралельних прямих завжди паралельні
* Довжини проекцій однакових відрізків паралельний прямих однакові, незалежно від їх віддаленості від площини проектування

*Основні застосування аксонометричної проекції:*

* Архітектне та технічне креслення
* Дизайн продуктів
* 3D моделювання для відеоігор тощо

*Перспективна проекція* — це лінійна проекція, де тривимірні об’єкти проектуються на площину зображення. Це призводить до того, що віддалені об’єкти здаються меншими, ніж ближні. Це також означає, що лінії, які є паралельними за своєю природою (тобто зустрічаються в точці на нескінченності), здаються перетинаними на проектованому зображенні, наприклад, якщо залізниці зображені з перспективною проекцією, вони здаються сходяться до однієї точки, яка називається точка зникнення. Фотооб’єктиви та людське око працюють однаково, тому перспективна проекція виглядає найбільш реалістичною. Тому перспективні проекції застосовують для створення зображень, наближених до того, як бачить око людини.

*Особливості перспективної проекції:*

* Не зберігається співвідношення довжин і площин
* Прямі лінії зображуються прямими лініями
* Паралельні прямі, паралельні площині екрану зображуються прямими, що сходяться в одній точці
* Паралельні прямі, непаралельній площині екрану, зображуються прямими, що сходяться в одній точці
* Довжини проекцій однакових відрізків будуть різними в залежності від їх віддаленості від площини проектування

*Основні застосування перспективної проекції:*

* *У архітектурі: для створення реалістичних зображень будівель з ціллю візуалізувати сприйтливо для людського ока.*
* *Дизайн інтер'єру*
* *У мистецтві (малювання)*
* *У фотографії: перспективи дають відчуття глибини та масштабності зображень*
* *Графічний дизайн*
* *тощо*

**8) Проектування 3D об'єктів на 2D площину. Який алгоритм найпоширеніший ? Який краще ?**

Звісно, вибір алгоритму залежить від області застосувань.

Проте часто найращим та найпоширенішим виявляється Z-буферизація. Вона реалізується як на апаратному рівні, так і в програмному забезпеченні.

Переваги: просто розробити; приглушує проблему швидкості, якщо здійснено в hardware; обробляє об'єкти по одному.

Недоліки: вимагає досить багато пам'яті; процес z-буферизації може бути часозатратним.

Існують вирішення цих недоліків:

- Метод Z-Compression - стиснення й розпакування зображення буферу

- Метод Fast Z-Clear швидке очищення Z-буфера. Після побудови і виведення зображення на екран, інформація, що розміщується у Z-буфері, вже неактуальна тому стирається. Z-буфер обнуляється, але вже не записуються окремі значення, а використовуються блоки які заміщають кілька значень одразу.

**Алгоритм Робертса 1963 р**

У алгоритмі Робертс розглядає тільки відрізки, що є перетином граней багатогранника. З узагальненої матриці можна отримати інформацію про те, які грані багатогранника перетинаються у вершинах.

**Warnock and Weiler-Atherton Area Subdivision**

Ділить зображення на менші і менші прямокутники, поки кожна область не є проекцією частини однієї видимої поверхні чи поверхні взагалі.

Цей процес продовжують до тих пір, поки можна віднести належність до однієї поверхні, або поки вони зменшуються до розміру одного пікселя.

**Алгоритм художника**

Алгоритм сортує всі багатокутники сцени по їх глибині і потім малює їх у порядку від найвіддаленішого до найближчого.

**Недолік:** Алгоритм, в деяких випадках, не може розташувати об'єкти.

Зокрема при циклічному перекритті або обробці багатокутників.

**Алгоритм сортування Depth Sorting Method**

Впорядкувати всі багатокутники з найменшою Z координатою (найвіддаленіша)

Вирішити невідповідності, які викликають накладання Z координати

Намалювати багатокутники в порядку зростання найменшої Z координати

**Алгоритм Невілла - “розрізання” багатокутників Newell-Newell-Sancha**

Для ліквідації циклу з багатокутників при сортування по глибині, який використовується для видалення невидимих поверхонь. Якщо немає сортування і не вдалося циклічно перекрити многокутники, то у цьому випадку, вони повинні бути виділені і сортування буде продовжене з нециклічних частин. Поділ відбувається на одному з многокутників на розрізаній кромці з іншого многокутника.

**Зворотній алгоритм художника**

виводить впорядковані об'єкти починаючи з найближчого закінчуючи найвіддаленішим за правилом, що фарба не повинна бути застосована до тої частини зображення, яка вже пофарбована. При побудові зображення, це може бути дуже ефективним, для частин більш віддалених від сцени, які закриті ближчими об'єктами.

**Недолік:** Алгоритм, в деяких випадках, не може розташувати об'єкти.

Зокрема при циклічному перекритті або обробці багатокутників.

**Глибина буфера (Z-буфер) Метод**

Спирається на глибину елементів зображення. Зазвичай реалізується на апаратному рівні, іноді в програмному забезпеченні - є попіксельним узагальненням алгоритму художника.

**Алгоритм Scan Line «рядок за рядком» Watkins**

**Перевага**

немає необхідності копіювати координати всіх вершин — лише вершини, що потрапляють в зону видимості.

Кожна з вершин зчитується лише раз, що значно підвищує швидкість виконання.

**Binary Space Partition (BSP) Trees**

Метод двійкового розбиття простору.

Перетворює вхідний список багатокутник в cтруктуру дерева BSP дерево.

**9) Опишіть основні крокі виконання завдання побудови 3D моделі з обертанням та освітленням.**

1. Створити 3D модель.

2. Зчитуємо (трикутні) грані модель

3. Для кожної точки грані робимо відповідний поворот (множення на матрицю повороту)

4. Вираховуємо інтенсивність освітлення кожної точки грані, як поверхні (відносно зафіксованого джерела освітлення): метод Гуро, метод Фонга,

5. Вираховуємо накладання текстур (якщо необхідно): Texture Mapping

6. Зображаємо зафарбовану грань (кольори визначені освітленням та з текстурою визначеною на попередньому кроці)

7. Повторюємо дії 3-7 для кожної грані.

**10) Основні кроки та функції виконання застосування в OpenGL.**

Функції:

• Функції опису примітивів визначають об'єкти нижнього рівня ієрархії (примітиви). Примітиви: точки, лінії, багатокутники і т.д.

• Функції опису джерел світла - для опису положення і параметрів джерел світла, розташованих у тривимірній сцені.

• Функції завдання атрибутів. За допомогою атрибутів визначається відображення на екрані: колір, характеристики матеріалу, текстури, параметри освітлення

• Функції візуалізації дозволяють задати положення спостерігача, параметри об'єктива камери для побудови зображення (в полі зору).

• Набір функцій геометричних перетворень - поворот, перенесення, масштабування.

Кроки:

• Ініціалізації вікна, налаштування функцій з зворотним викликом і головного циклу обробки подій.

• Ініціалізація вікна складається з налаштування відповідних буферів кадру, початкового положення і розмірів вікна, а також заголовка вікна.

• Функція glutInit (& argc, argv) ініціалізацію бібліотеки GLUT.

• Команда glutInitDisplayMode (GLUT\_RGB) ініціалізує буфер кадру і налаштовує повноколірний (непалітровий) режим RGB.

• glutInitWindowSize (Width, Height) використовується для завдання початкових розмірів вікна.

• glutCreateWindow ( "Glut Shapes ") задає заголовок вікна і візуалізує саме вікно на екрані.

• функції Display (), Reshape () і Keyboard () функції, викликаються відповідно, при перемальовуванні вікна, зміні розмірів вікна, та натисканні клавіші на клавіатурі.

• Контроль за усіма подіями і виклик потрібних функцій відбувається всередині нескінченного циклу в функції glutMainLoop ().