Лабораторна робота №4. Моделювання освітлення

Мета: отримати навички програмування освітлення тривимірних об’єктів засобами графіки OpenGL ES.

У цій лабораторній роботі будуть розглянуті питання програмування рендерингу тривимірних об’єктів з урахуванням основних законів та властивостей освітлення.

Завдання

Потрібно створити у середовищі Android Studio проєкт з ім’ям **Lab4_GLES**, зокрема

- написати вихідний текст програми згідно варіанту завдання. Використати мову Java або Kotlin – на вибір
- Налагодити програму. Перевірити роботу програми на емуляторі та на фізичному пристрої Android.

Засоби для виконання лабораторної роботи

Android Studio – середовище розробки Android-застосунків.

Теоретичні положення та методичні рекомендації

Створення проєкту Android Studio – так само, як і для попередніх лабораторних робіт. Варто нагадати, що для успішного використання OpenGL ES версії 3.2 потрібно вказати цільовий API рівня не нижче 28 (Android 9.0).

Далі потрібно розглянути основні положення щодо моделювання відбиття світла, зокрема, основні закономірності, властивості та відповідні моделі.

Модель відбиття світла

Узагальнено сумарна інтенсивність освітлення кожної точки поверхні будь-якого об'єкта сцени на основі врахування різних факторів відбиття світла може бути емпірично виражена наступною формулою:

$$I_{res} = I_a K_a + I_d K_d + I_s K_s,$$

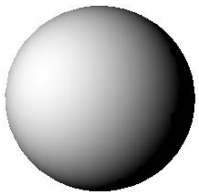
де:

I_a – інтенсивність фонового (*ambient*) підсвічування,

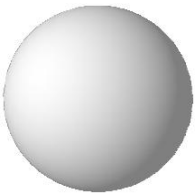
I_d – інтенсивність дифузного (*diffuse*) відбиття світла,

I_s – інтенсивність дзеркального (*specular*) відбиття світла,

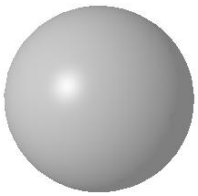
K_a , K_d , K_s – параметри, які описують матеріали, з яких зроблені об'єкти та інші показники.



Тільки дифузне
$K_a = 0, K_d = 1, K_s = 0$



Дифузне та фонове
$K_a = 0.5, K_d = 0.5, K_s = 0$

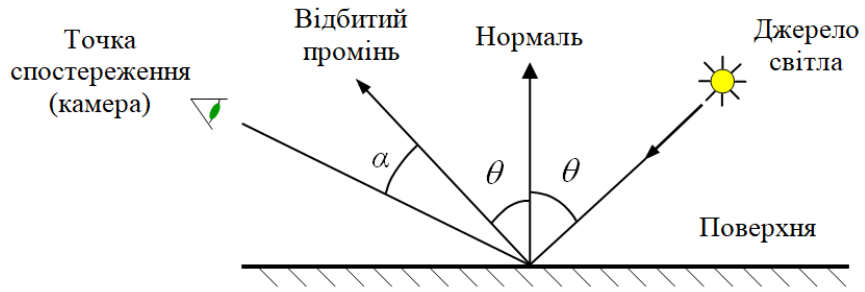


Наявні усі три складові
$K_a = 0.5, K_d = 0.3, K_s = 0.2$

Ілюстрація трьох складових моделі відбиття світла

Розглянемо докладніше окремі складові моделі відбиття світла.

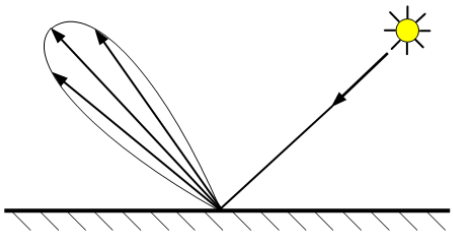
Для дзеркального відбиття кут θ між нормаллю і падаючим променем дорівнює куту між нормаллю і відбитим променем. Падаючий промінь від джерела світла, відбитий промінь і нормаль розташовуються в одній площині



Дзеркальне відбиття світла

Падаючий промінь, потрапляючи на шорсткувату поверхню реального дзеркала, породжує не один відбитий промінь, а кілька променів, що

розсіюються по різних напрямках. Зона розсіювання залежить від якості полірування і може бути описана деяким законом розподілу. Як правило, форма зони розсіювання симетрична щодо лінії ідеального дзеркально відбитого променя. Це можна відобразити такою векторною діаграмою



До числа найпростіших, але досить часто уживаних, відноситься емпірична залежність, запропонована *Фонгом*, відповідно до якої інтенсивність (I_s) дзеркально відбитого випромінювання світла пропорційна $(\cos\alpha)^p$, де α – кут відхилення від лінії ідеально відбитого променя. Запишемо це у такий спосіб:

$$I_s = I\cos^p\alpha\,,$$

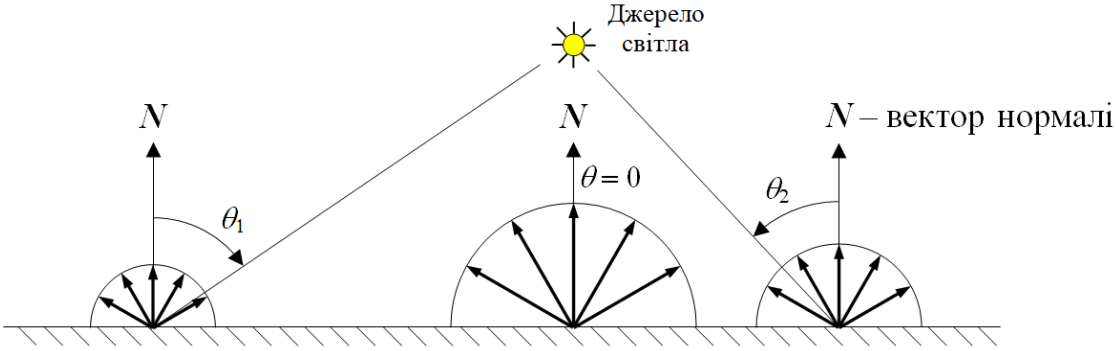
де I – інтенсивність випромінювання джерела світла. Показник розсіювання p знаходиться в діапазоні від 1 до 1000 і певною мірою описує якість полірування поверхні.

Дифузне відбиття світла. Цей вид відбиття притаманний *матовим* поверхням. Матовою можна вважати таку поверхню, розмір шорсткості якої вже настільки великий, що падаючий промінь розсіюється рівномірно в усі сторони. Такий тип відбиття світла характерний, наприклад, для гіпсу, піску, паперу.

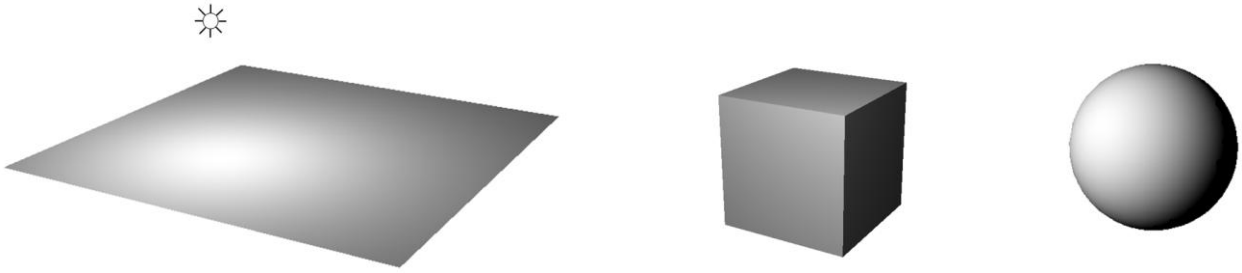
Дифузне відбиття світла описується *законом Ламберта*, згідно з яким інтенсивність відбитого світла (I_d) пропорційна косинусу кута θ між напрямом на точкове джерело світла й нормаллю до поверхні

$$I_d = I\cos\theta\,,$$

де I – інтенсивність джерела світла



Дифузне відбиття світла



Приклади дифузного відбиття світла від точкового джерела

Таким чином, вираження для суми трьох компонент моделі відбиття світла

$$I_{res} = I_a K_a + I_d K_d + I_s K_s$$

можна переписати з урахуванням виражень для дифузної та дзеркальної компонент

$$I_{res} = I_a K_a + I (K_d \cos\theta + K_s \cos^p\alpha),$$

Таку трьохкомпонентну модель відбиття світла називають моделлю Фонга для відбиття світла (*Phong reflection model* або *Phong lighting model*).

Якщо врахувати те, що енергія від джерела світла зменшується відповідно відстані до нього, то це можна виразити так

$$I_{res} = I_a K_a + I \cdot F(D) \cdot (K_d \cos\theta + K_s \cos^p\alpha),$$

де D – відстань від джерела світла до поверхні, $F(D)$ – функція ослаблення (*attenuation*). Для точкового джерела світла енергія випромінювання зменшується пропорційно квадрату відстані.

Частковим випадком можна назвати варіант, коли маємо єдине джерело світла. У цьому випадку можна вважати, що внесок фонової компоненти ($I_a K_a$) також є пропорційним функції відстані до джерела. Тоді можна записати так

$$I_{res} = F(D) \cdot (I_a K_a + I \cdot (K_d \cos\theta + K_s \cos^p\alpha))$$

Для декількох джерел світла дифузні та дзеркальні складові розраховують окремо для кожного джерела, а результат є сумою усіх складових:

$$I_{res} = I_a K_a + K_d \sum_j I_d(j) + K_S \sum_j I_s(j).$$

Як використати надану модель відбиття для розрахунку кольору зафарбовування точок об'єктів? Якщо маємо кольорові джерела світла, що освітлюють кольорові поверхні, тоді запишемо формулу для кольору результату у наступному вигляді

$$C_{res} = C_a\, C_p\, K_a \; + C\, C_p\, K_d \cos\,\theta + C\, K_s \cos^{p}\alpha\,,$$

де C – колір джерела світла, C_p – колір поверхні об’єкта, C_a – колір фонового підсвічування. Для дзеркальної складової колір відбиття не залежить від власного кольору поверхні, він визначається тільки кольором джерела освітлення.

Перепишемо цю формулу для компонент кольорової моделі *RGB*:

$$\begin{pmatrix} R_{res} \\ G_{res} \\ B_{res} \end{pmatrix} = \begin{pmatrix} R_a \\ G_a \\ B_a \end{pmatrix} \begin{pmatrix} R_p \\ G_p \\ B_p \end{pmatrix} K_a \; + \begin{pmatrix} R \\ G \\ B \end{pmatrix} \begin{pmatrix} R_p \\ G_p \\ B_p \end{pmatrix} K_d \cos\,\theta \; + \begin{pmatrix} R \\ G \\ B \end{pmatrix} K_s \cos^{p}\alpha$$

Прокоментуємо цю формулу. Компоненти R_p, G_p та B_p кольору поверхні тут помножуються на відповідні компоненти кольору світла для фонової та дифузної складових. Колір результату освітлення матової поверхні дорівнює добутку компонентів кольору поверхні та компонентів кольору джерела світла. Наприклад, якщо жовту матову поверхню освітлювати блакитним джерелом світла, то відповідно моделі дифузного відбиття світла поверхня буде виглядати зеленою:

$$\begin{pmatrix} R_{res} \\ G_{res} \\ B_{res} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

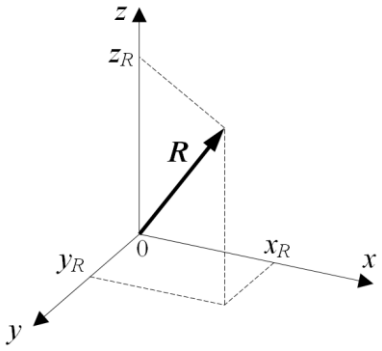
Для дзеркальної складової колір відбиття не залежить від власного кольору поверхні, він визначається тільки кольором джерела освітлення.

Деякі відомості щодо алгебри векторів

Математичний апарат алгебри векторів широко застосовується в моделях освітлення та програмуванні комп'ютерної графіки.

Вектором називається відрізок прямої, що з'єднує деякі точки простору *A* і *B*. Напрямок вектора – від початкової точки *A* до точки *B*.

Радіус-вектор ***R*** – це вектор із початковою точкою в центрі координат. Координатами радіуса-вектора є координати кінцевої точки (x_R, y_R, z_R).



Довжина радіуса-вектора часто називається ***модулем***, позначається як |***R***| і обчислюється в такий спосіб:

$$\left|\mathbf{R}\right| = \sqrt{x_R^2 + y_R^2 + z_R^2}\,.$$

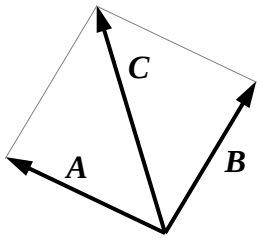
Одиничний вектор – це вектор, довжина якого дорівнює одиниці.

Оглянемо основні операції над векторами.

1. Множення вектора на число. Таку операцію можна записати як ***U*** = ***Va***. Результат – вектор ***U***, довжина якого в *a* раз більша, ніж у вектора ***V***. Якщо число *a* позитивне, то напрямок вектора ***U*** збігається з вектором ***V***. При *a<0* вектор ***U*** має напрямок протилежний вектору ***V***. Якщо ***V*** – це радіус-вектор, то координати вектора результату будуть *(a·x, a·y, a·z).*

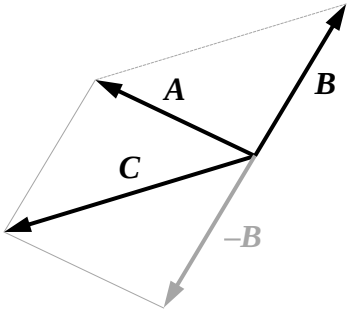
2. **Додавання векторів *C* = *A* + *B***. Результат додавання – це вектор, що відповідає одній з діагоналей паралелограма, сторонами якого є вектори ***A*** і ***B***. Усі три вектори лежать в одній площині. Для радіусів-векторів ***A*** і ***B*** координати радіус-вектора результату ***C*** визначаються так:

$$x_C = x_A + x_B$$
$$y_C = y_A + y_B$$
$$z_C = z_A + z_B$$



Віднімання двох векторів $\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$ можна визначити через операцію додавання $\boldsymbol{C} = \boldsymbol{A} + (-\boldsymbol{B})$. Вектор різниці відповідає діагоналі паралелограма зі сторонами $\boldsymbol{A}$ та $(-\boldsymbol{B})$. При відніманні радіусів-векторів відповідні координати віднімаються:

$$x_C = x_A - x_B$$
$$y_C = y_A - y_B$$
$$z_C = z_A - z_B$$



3. **Скалярний добуток векторів** $c = \boldsymbol{A}\cdot\boldsymbol{B}$. Результатом операції є число (скаляр), що дорівнює добутку довжин векторів на косинус кута між ними:

$$c = \boldsymbol{A}\cdot\boldsymbol{B} = |\boldsymbol{A}|\,|\boldsymbol{B}|\cos\varphi.$$

Якщо $\boldsymbol{A}$ і $\boldsymbol{B}$ – це радіуси-вектори, то результат можна обчислити через координати в такий спосіб

$$\mathrm{c} = x_A x_B + y_A y_B + z_A z_B\,.$$

Англомовна назва скалярного добутку векторів – *dot product*.

4. **Векторний добуток векторів** $\boldsymbol{C} = \boldsymbol{A}\times\boldsymbol{B}$. Результатом операції є вектор, перпендикулярний до площини паралелограма, утвореного сторонами векторів $\boldsymbol{A}$ і $\boldsymbol{B}$, а довжина вектора дорівнює площі цього паралелограма

$$|\,\boldsymbol{C}\,| = |\boldsymbol{A}|\,|\boldsymbol{B}|\sin\varphi.$$

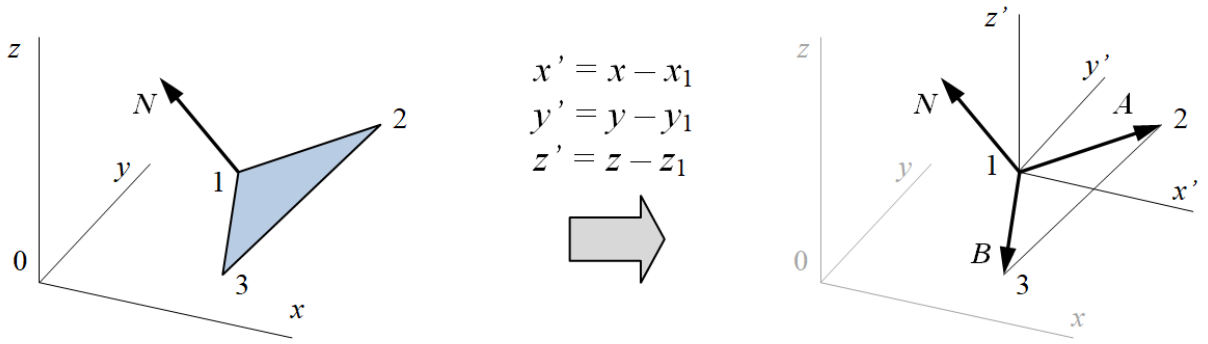
Докладніше про векторний добуток див. лекційний матеріал.

Англомовна назва скалярного добутку векторів – *cross product*.

Знаходження векторів нормалей

Нехай у просторі задана деяка багатогранна поверхня. Розглянемо одну її плоску грань у вигляді трикутника. Потрібно знайти координати вектора нормалі $\boldsymbol{N}$.

Для обчислення координат вектора нормалі скористаємося векторним добутком будь-яких двох векторів, що лежать у площині грані. Такими векторами можуть служити й ребра грані, наприклад, ребра 1-2 і 1-3. Однак формули для векторного добутку були визначені нами тільки для радіусів-векторів. Щоб перейти до радіусів-векторів, уведемо нову систему координат, центр якої збігається з вершиною 1, а осі паралельні осям колишньої системи, як проілюстровано на рисунку нижче.



Для знаходження нормалі потрібної сторони площини трикутника необхідно враховувати, яке розташування осей координат та який порядок нумерації вершин трикутника. Для знаходження координат вектора нормалі при індексації вершин по годинниковій стрілці, у *правій* системі координат потрібно використовувати наступні формули

$$x'_N = (z_2 - z_1)(y_3 - y_1) - (y_2 - y_1)(z_3 - z_1)$$
$$y'_N = (x_2 - x_1)(z_3 - z_1) - (z_2 - z_1)(x_3 - x_1)$$
$$z'_N = (y_2 - y_1)(x_3 - x_1) - (x_2 - x_1)(y_3 - y_1)$$

Примітка. Права система координат використовується у OpenGL.

Якщо полігональна поверхня має не трикутні грані, а, наприклад, плоскі чотирикутні, то розрахунок нормалі можна виконувати по будь-яких трьох вершинах. Головне – щоб ці три вершини не були на одній прямій.

Розрахунки для дифузного відбиття світла

Потрібно знайдемо косинус кута між вектором нормалі і напрямком на джерело світла. Це можна виконати у такій спосіб. Спочатку необхідно визначити радіус-вектор, спрямований на джерело світла. Позначимо його як $\boldsymbol{S}$. Потім, для обчислення косинуса кута між радіусами-векторами $\boldsymbol{S}$ і $\boldsymbol{N}$ скористаємося формулами скалярного добутку векторів. Позаяк

$$\boldsymbol{S} \cdot \boldsymbol{N} = |\boldsymbol{S}|\,|\boldsymbol{N}| \cos\theta\,,$$

у координатах це

$$\boldsymbol{S} \cdot \boldsymbol{N} = x_S x_N + y_S y_N + z_S z_N\,,$$

то дістанемо

$$\cos\theta = \frac{x_S x_N + y_S y_N + z_S z_N}{|\boldsymbol{S}||\boldsymbol{N}|}$$

Вочевидь для спрощення обчислень доцільно використовувати радіус-вектори $\boldsymbol{S}$ та $\boldsymbol{N}$ одиничної довжини, тобто $|\boldsymbol{S}|\,|\boldsymbol{N}| = 1$. Тоді косинус кута дорівнює скалярному добутку цих векторів: $\cos\theta = x_S x_N + y_S y_N + z_S z_N$.

Методичні рекомендації щодо програмування освітлення

Щодо формату масиву вершин

Потрібно вирішити наступні питання:
- де та як зберігати нормалі?
- де та як зберігати кольори об’єктів?

Можливі, як мінімум, два варіанти:

1. У форматі массиву вершин для кожної вершини окрім координат (*x*,*y*,*z*) записувати кольори (*R, G, B*) та координати нормалей (x_N, y_N, z_N). Разом це виходить по вже 9 значень типу float на кожну вершину. Це збитковий варіант з точки зору витрат пам'яті, але у нього є позитивна риса – уся інформація про об'єкти одразу записуються у пам'ять даних для графічного процесора. Це може забезпечити високу швидкість рендерингу, особливо у режимі анімації.

2. Інший варіант – у масиві вершин для кожної вершини зберігати разом лише координати вершин (*x*,*y*,*z*) та координати нормалей (x_N, y_N, z_N). Якщо не зберігати кольори у масиві вершин, то можна (і такий варіант детально розглядався у лаб. роботі №2) завантажувати індивідуально поточний колір для об'єктів через uniform-перемінну шейдера фрагментів перед відповідними викликами glDrawArrays() у циклі рендерингу. Це збільшує навантаження на центральний процесор, але може бути прийнятним у багатьох конкретних випадках.

У лабораторній роботі №4 пропонується використати останній варіант. Формат масиву вершин можна відобразити так

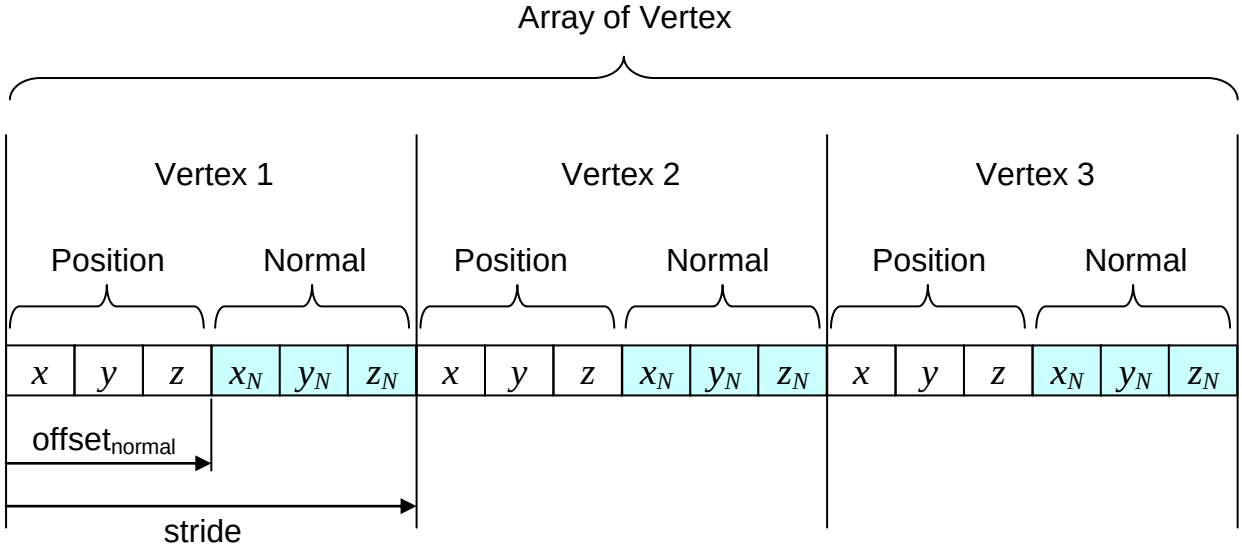


Рис. 1. Поєднання координат вершин та нормалей у єдиному масиві вершин

Розглянемо приклад визначення кольорів об'єктів відповідно обраному способу. Введемо у фрагментний шейдер uniform-перемінну з ім’ям, наприклад, vColor

```
precision mediump float;
uniform vec4 vColor;
out vec4 resultColor;

void main() {
   resultColor = vColor;
}
```

А далі вже у циклі рендерингу (під час роботи метода onDrawFrame) через перемінну vColor фрагментному шейдеру вказувати поточний колір для відповідних граней об'єктів

```
int uHandle = GLES32.glGetUniformLocation(gl_Program, "vColor");
GLES32.glUniform3fv(uHandle, 1, objectColor, 0);
GLES32.glDrawArrays(. . .);       //draw something
```

Шаблон програмного коду

Основа цього шаблону – файли MainActivity.java та myWorkMode.java. Пропонується використати той самий шаблон програмного коду, що і для попередньої лаб. №3, який було названо Шаблон 5. І це незважаючи на те, що у даній лаб. роботі змінено формат масиву вершин.

У шаблоні програмного коду передбачено методи, які виконують компіляцію шейдерів та завантаження масива вершин.

```
public void myCreateShaderProgram() {
    myCompileAndAttachShaders(
            myShadersLibrary.vertexShaderCode6,
            myShadersLibrary.fragmentShaderCode6);

    myVertexArrayBind2(arrayVertex, 6,
            "vPosition", 0,
            "vNormal", 3*4);
}
```

В аргументах метода myVertexArrayBind2 вказані імена вхідних перемінних **vPosition** та **vNormal** – вони будуть означати атрибути координат та нормалей у масиві вершин, з якими буде працювати шейдер вершин.

Шаблон шейдера вершин

```
#version 300 es
in vec3 vPosition;
in vec3 vNormal;

uniform mat4 uModelMatrix;
uniform mat4 uViewMatrix;
uniform mat4 uProjMatrix;

out vec3 currentPos;
out vec3 currentNormal;

void main() {
   gl_Position = uProjMatrix*uViewMatrix*uModelMatrix*vec4(vPosition, 1.0f);
   currentPos = mat3(uModelMatrix) * vPosition;
   currentNormal = mat3(uModelMatrix) * vNormal;
}
```

Вектори currentPos та currentNormal мають враховувати можливі повороти об'єктів, які можуть визначатися матрицею uModelMatrix.

Значення перемінних currentPos та currentNormal будуть сприйматися у фрагментному шейдері, який буде використовувати ці значення для обчислення компонентів моделі освітлення.

Шаблон фрагментного шейдера

```
#version 300 es
precision mediump float;
in vec3 currentPos;
in vec3 currentNormal;

uniform vec3 vColor;
uniform vec3 vLightPos;

out vec4 resultColor;

void main() {
    vec3 norm = normalize(currentNormal);
    vec3 lightDir = normalize(vLightPos - currentPos);

    //. . . do something

    resultColor = vec4(.., 1.0f);
}
```

Далі розглянемо основні моменти програмування фрагментних шейдерів для реалізації певних різновидів моделі відбиття світла.

Шейдери дифузного відбиття світла

Розглянемо, як можна запрограмувати фрагментний шейдер для *ambient* + *diffuse* компонент згідно такої моделі

$$C_{res} = C_p K_a + C C_p K_d \cos \theta,$$

де C – колір джерела світла, C_p – колір поверхні об'єкта, θ – кут між напрямом на точкове джерело світла й нормаллю до поверхні, K_a та K_d – вагові коефіцієнти відповідно для *ambient* та *diffuse* компонент.

```
#version 300 es
precision mediump float;
in vec3 currentPos;
in vec3 currentNormal;

uniform vec3 vColor;
uniform vec3 vLightColor;
uniform vec3 vLightPos;

out vec4 resultColor;

void main() {
    vec3 norm = normalize(currentNormal);
    vec3 lightDir = normalize(vLightPos - currentPos);
    float diffuse = max(dot(norm, lightDir), 0.0);
    vec3 vClr = 0.3f*vColor + 0.7f*diffuse*vLightColor*vColor;
    resultColor = vec4(vClr, 1.0f);
}
```

Спочатку нормалізуються вектори – вектор поточної нормалі до поверхні грані та вектор напрямку на джерело світла відносно поточної точки поверхні (vLightPos - currentPos) цієї ж грані. Як вже вказувалося вище, значення векторів currentPos та currentNormal формуються у шейдері вершин на основі значень координат вершин та координат векторів нормалей у вершинах.

Косинус кута (θ) між нормаллю та напрямком на джерело світла обчислюється як dot(norm, lightDir) і записується у перемінну diffuse. Для того, щоб відкинути від'ємні значення косинуса використана функція max().

Далі виконується множення трьохкомпонентних (RGB) векторів кольорів vColor (колір поверхні) та vLightColor (колір джерела світла) на коефіцієнти $K_a = 0.3$ та $K_d = 0.7$ та на косинус кута θ. Поточний колір точки фрагменту записується у вихідну перемінну resultColor.

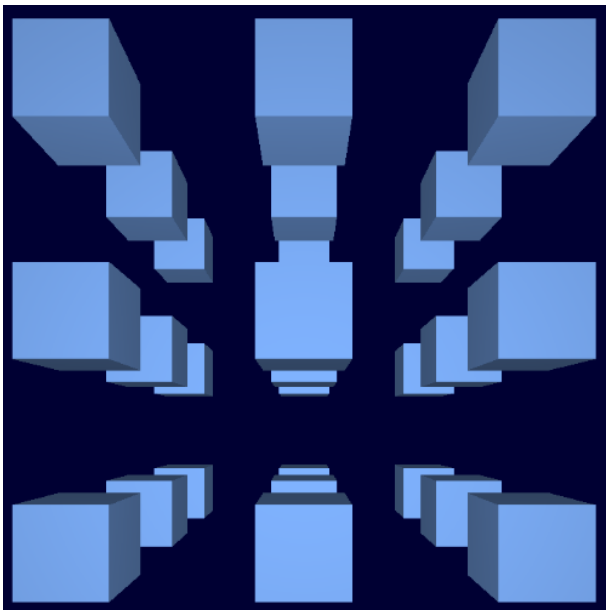
Такий шейдер можна віднести до найпростіших варіантів щодо моделювання дифузного відбиття та фонового підсвічування.

Більш досконалим з позицій створення ілюзій реалістичного зображення є врахування зменшення енергії від джерела світла пропорційно

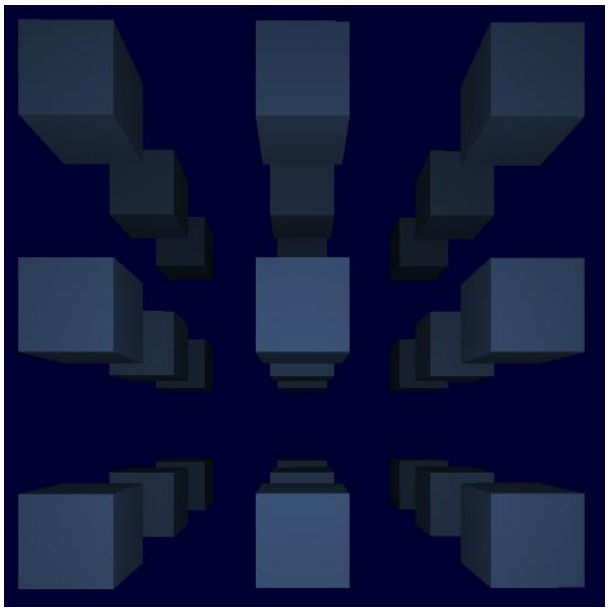
відстані (а точніше – квадрату відстані). Нижче наведено шейдер, який втілює такий підхід

```
#version 300 es
precision mediump float;
in vec3 currentPos;
in vec3 currentNormal;

uniform vec3 vColor;
uniform vec3 vLightColor;
uniform vec3 vLightPos;

out vec4 resultColor;

void main() {
    vec3 norm = normalize(currentNormal);
    vec3 lightDir = normalize(vLightPos - currentPos);
    float distance = length(vLightPos - currentPos);
    float attenuation = 1.0f/(1.0f + distance*distance);
    float diffuse = max(dot(norm, lightDir), 0.0);
    vec3 vClr = 0.3f*vColor + 0.7f*diffuse*vLightColor*vColor;
    resultColor = vec4(attenuation *vClr, 1.0f);
}
```

Порівняння роботи шейдерів дифузного відбиття свіла



Освітлення без урахування відстані



Освітлення зменшується при віддаленні від джерела світла

Примітка. Для написання шейдерів до цієї лаб. значною мірою було використано джерело:

Joey de Vries. LearnOpenGL. https://learnopengl.com/book/book_pdf.pdf на сайті https://learnopengl.com/

Шейдер дзеркального відбиття світла

Розглянемо, як можна запрограмувати фрагментний шейдер для *ambient* + *specular* компонент згідно такої моделі

$$I_{res} = C_p K_a + C K_s \cos^p \alpha ,$$

де C – колір джерела світла, C_p – колір поверхні об'єкта, α – кут відхилення напряму зору від лінії ідеально відбитого променя, p – показник розсіювання, K_a та K_s – вагові коефіцієнти відповідно для *ambient* та *specular* компонент.

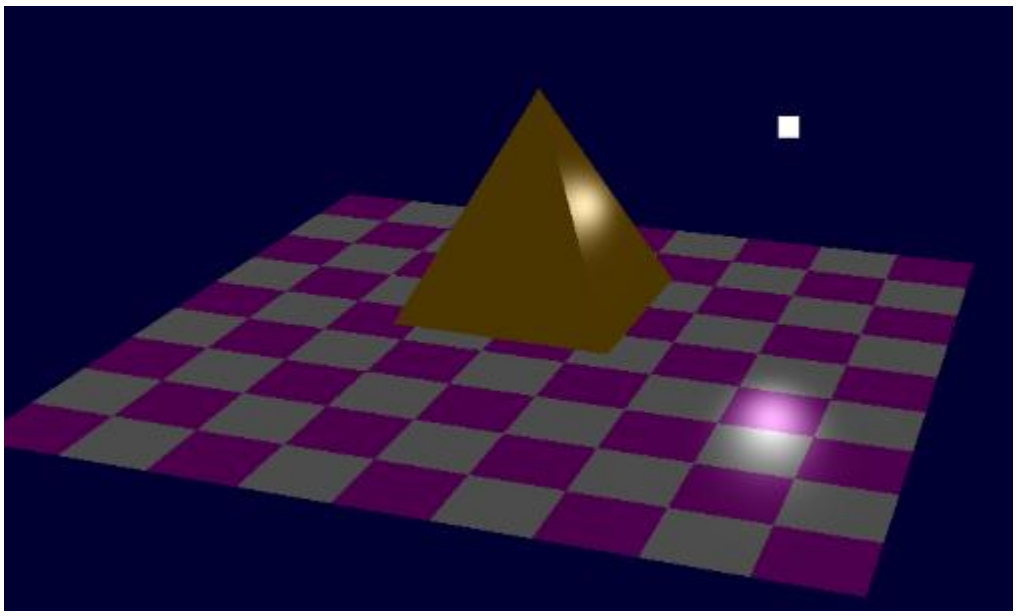
```
#version 300 es
precision mediump float;
in vec3 currentPos;
in vec3 currentNormal;

uniform vec3 vColor;
uniform vec3 vLightColor;
uniform vec3 vLightPos;
uniform vec3 vEyePos;

out vec4 resultColor;

void main() {
    vec3 norm = normalize(currentNormal);
    vec3 lightDir = normalize(vLightPos - currentPos);
    vec3 reflectDir = normalize(reflect(-lightDir, norm));
    vec3 eyeDir = normalize(vEyePos - currentPos);
    float spec = max(dot(eyeDir, reflectDir), 0.0f);
    spec = pow(spec, 200.0f);
    vec3 vClr = 0.3f*vColor + 0.7f*spec*vLightColor;
    resultColor = vec4(vClr, 1.0f);
}
```

Деякою мірою такий шейдер є подібним шейдеру дифузного відбиття світла – це стосується формування вектора нормалі та вектора напрямку на джерело світла.

Знаходження вектора відбитого проміня (reflectDir) у поточній точці поверхні грані виконуються за допомогою функції reflect(-lightDir, norm). Також знаходиться вектор напрямку від поточної точки до точки спостереження (камери) eyeDir = normalize(vEyePos - currentPos). Косинус кута відхилення напряму зору від лінії ідеально відбитого променя обчислюється як скалярний добуток векторів eyeDir та reflectDir. Результат записується у перемінну spec. Далі косинус кута підноситься у степінь 200 – це відповідає достатньо різким дзеркальним відблискам.

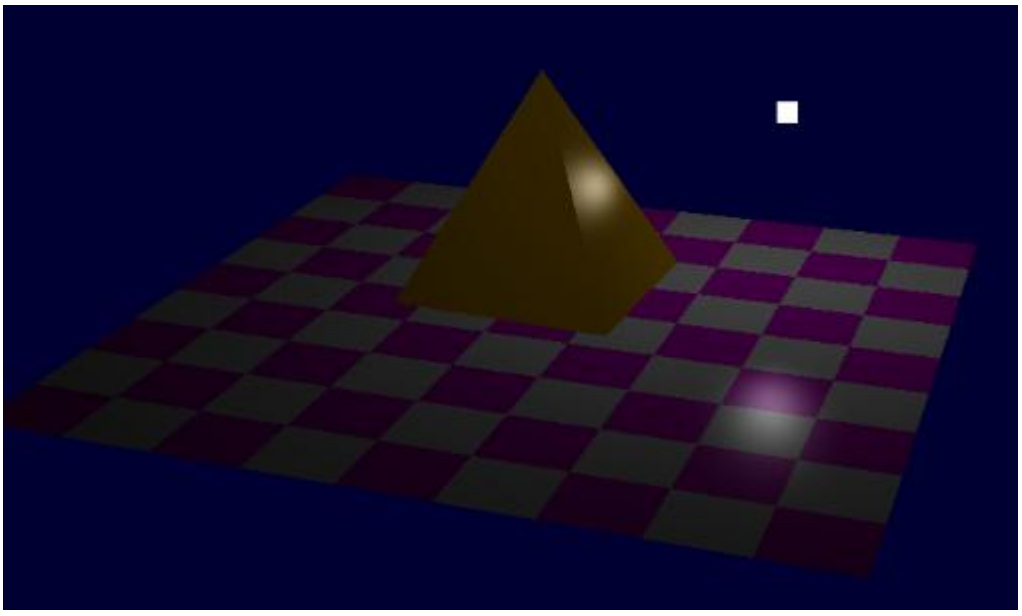
Колір результату складається з ambient-компоненти, помноженої на коефіцієнт 0.3, та дзеркальної компоненти из вагою 0.7.

Результат роботи такого шейдера наведено нижче.



Робота фрагментного шейдера для *ambient* + *specular* компонент

Подібно до того, як було це запроновано у шейдері дифузного відбиття світла, тут також можна ввести функцію зменшення освітлення пропорційно квадрату відстані від точкового джерела світла. Можна прямо використати рядки коду для обчислення величини atten і вставити це у шейдер дзеркального відбиття і помножити вектор кольору результату на atten. Результат проілюстровано нижче.



Тут освітлення кожної точки простору зменшується відповідно квадрату відстані до джерела світла

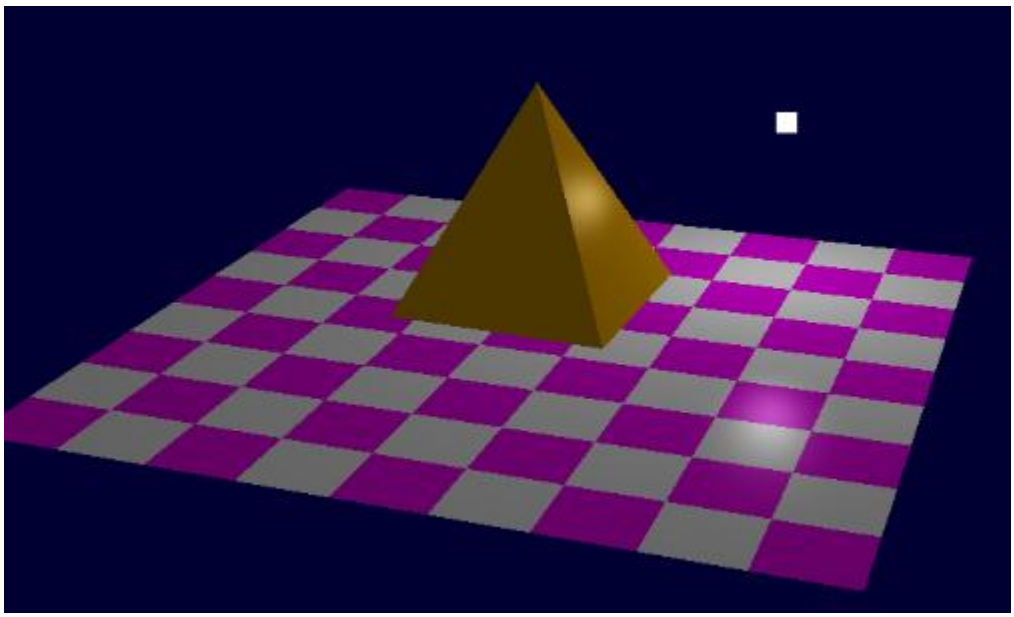
Шейдер дифузного та дзеркального відбиття світла

Розглянемо, як можна запрограмувати фрагментний шейдер для повної (*ambient + diffuse + specular)* моделі Фонга для відбиття світла

$$I_{res} = C_p K_a + C C_p K_d \cos\theta + C K_s \cos^p \alpha$$

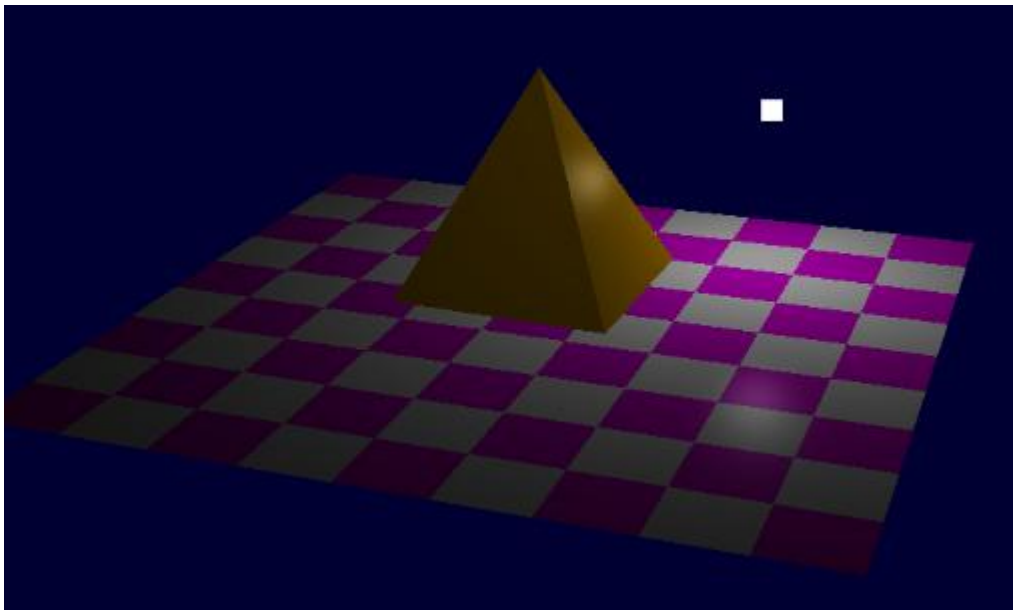
Програмний код шейдера для такого варіанту буде об'єднувати рядки коду попередніх шейдерів

```
#version 300 es
precision mediump float;
in vec3 currentPos;
in vec3 currentNormal;

uniform vec3 vColor;
uniform vec3 vLightColor;
uniform vec3 vLightPos;
uniform vec3 vEyePos;

out vec4 resultColor;

void main() {
   vec3 norm = normalize(currentNormal);
   vec3 lightDir = normalize(vLightPos - currentPos);
   float diffuse = max(dot(norm, lightDir), 0.0);

   vec3 reflectDir = normalize(reflect(-lightDir, norm));
   vec3 eyeDir = normalize(vEyePos - currentPos);
   float spec = max(dot(eyeDir, reflectDir), 0.0f);
   spec = pow(spec, 200.0f);

   vec3 vClr = 0.3f*vColor
             + 0.4f*diffuse*vLightColor*vColor
             + 0.3f*spec*vLightColor;
   resultColor = vec4(vClr, 1.0f);
}
```



Шейдер для 0.3**ambient* + *0.4***diffuse* + 0.3**specular*

Далі також можна ввести функцію зменшення освітлення пропорційно квадрату відстані від точкового джерела світла, як це було розглянуто у другому варіанті шейдеру дифузного відбиття світла. Якщо помножити вектор кольору результату на atten, то отримаємо наступний зезультат



Результат для attenuation*(0.3**ambient* + *0.4***diffuse* + 0.3**specular)*

Як показувати джерело світла?

Для того, щоб показувати точкове джерело світла, можна ввести у масив вершин одну точку (першу), наприклад, викликом функції

```
arrayVertex = new float[size];
int pos = 0;
pos = myGraphicPrimitives.addVertexXYZn(arrayVertex, pos, 0,0,0, 1,1,1);
```

А потім можна відображати таку точку викликом функції glDrawArrays(). Але як перемістити цю точку відповідно розташуванню джерела світла – наприклад, відповідно значень (x,y,z) масиву lightPos[3]? Для цього можна перед викликом glDrawArrays() сформувати матрицю перетворення об'єктів (modelMatrix) як матрицю зсуву на позицію світла

```
Matrix.setIdentityM(modelMatrix, 0);
Matrix.translateM(modelMatrix, 0, lightPos[0], lightPos[1], lightPos[2]);
uHandle = GLES32.glGetUniformLocation(gl_Program, "uModelMatrix");
GLES32.glUniformMatrix4fv(uHandle, 1, false, modelMatrix, 0);
GLES32.glDrawArrays(GLES32.GL_POINTS, 0, 1);
```

Для того, щоб точка відображалася достатньо помітною, можна у шейдері вершин записати gl_PointSize = 15.0f

Рекомендації щодо вводу примітивів у масив вершин

Для структуризації рішення можна визначити власний клас і назвати його, наприклад, **myGraphicPrimitives.** У ньому визначити методи для вводу потрібних графічних примітивів – трикутників, чотирикутників, пірамід, кубів тощо.

Рекомендується розпочати з визначення метода для вводу окремої вершини, наприклад, так

```
public static int addVertexXYZn(float[] vdest, int pos,
                                float x, float y, float z,
                                float xn, float yn, float zn) {
    float N = (float)Math.sqrt(xn*xn+yn*yn+zn*zn);
    if (N <= 0) return pos;
    vdest[pos++] = x;
    vdest[pos++] = y;
    vdest[pos++] = z;
    vdest[pos++] = xn/N;
    vdest[pos++] = yn/N;
    vdest[pos++] = zn/N;
    return pos;
}
```

Також у цьому методі корисно передбачити нормалізацію координат векторів нормалей.

Далі розглянемо приклад методу для уведення трикутних граней

```
public static int addTriangleXYZn(float[] vdest, int pos,
                                  float x1, float y1, float z1,
                                  float x2, float y2, float z2,
                                  float x3, float y3, float z3) {

    float xn = (z2-z1)*(y3-y1) - (y2-y1)*(z3-z1);
    float yn = (x2-x1)*(z3-z1) - (z2-z1)*(x3-x1);
    float zn = (y2-y1)*(x3-x1) - (x2-x1)*(y3-y1);
    pos = addVertexXYZn(vdest, pos, x1, y1, z1, xn, yn, zn);
    pos = addVertexXYZn(vdest, pos, x2, y2, z2, xn, yn, zn);
    pos = addVertexXYZn(vdest, pos, x3, y3, z3, xn, yn, zn);
    return pos;
}
```

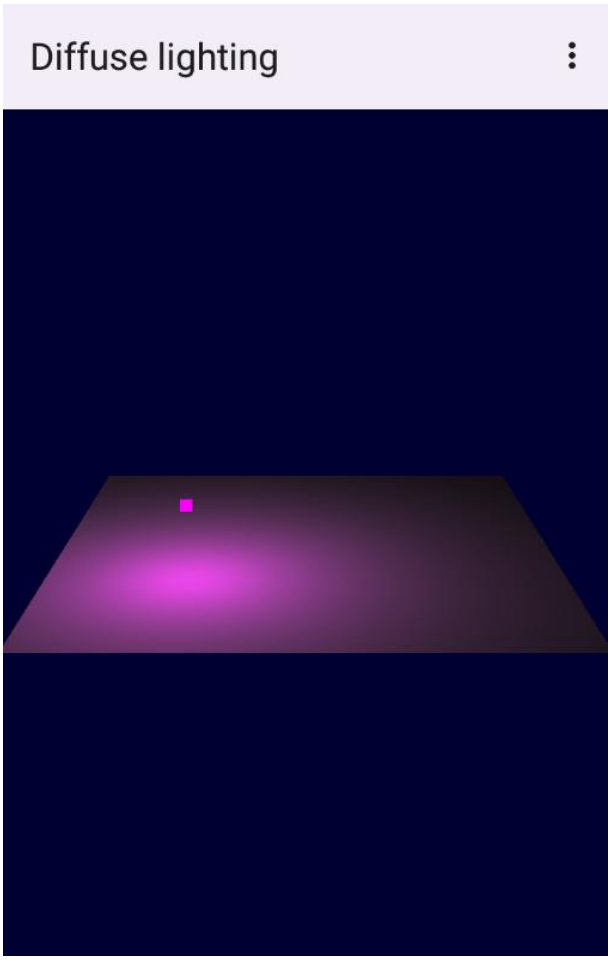
У цьому методі передбачено автоматичне обчислення координат нормалей вершин. Необхідно зазначити, що вершини трикутників потрібно вказувати порядку обходу за годинниковою стрілкою – тоді нормаль буде дивитися у відповідному напрямку (дивіться теоретичний матеріал щодо векторного добутку векторів).

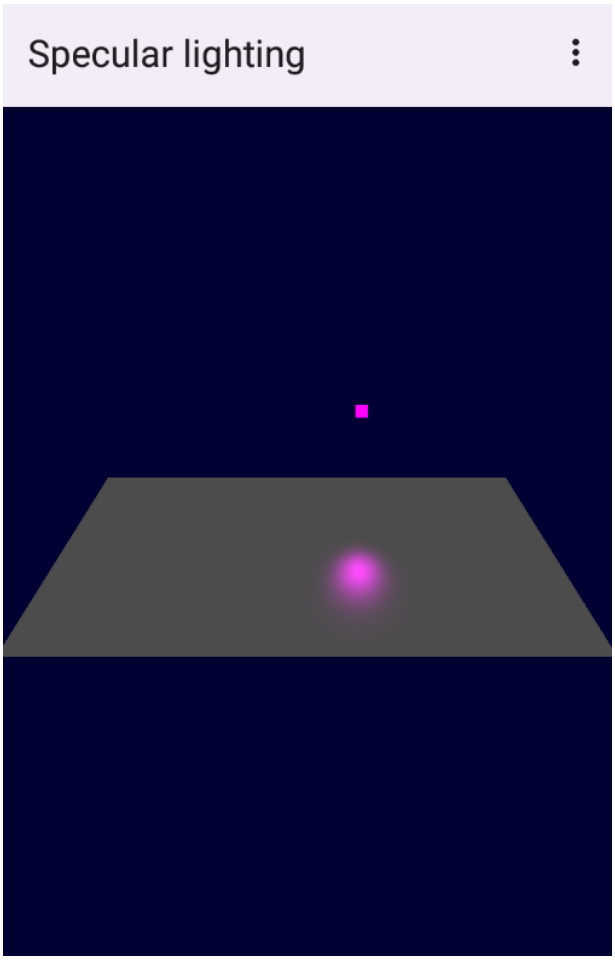
Далі можна визначити уведення чотирикутних граней з двох трикутників (у конфігурації TRIANGLE_FAN, або по-іншому).

На основі трикутників та чотирикутників можна програмувати уведення складніших об’єктів – піраміди, кубів, шахового поля тощо.

Варіанти завдань та основні вимоги

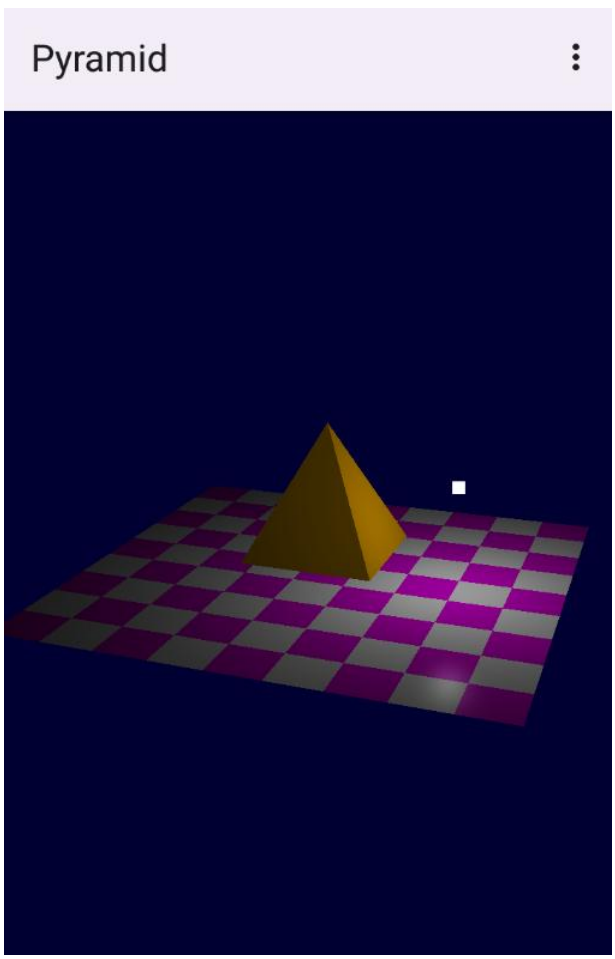
1. Застосунок **Lab4_GLES** для вибору режиму роботи повинен мати меню з двома пунктами:
    - Diffuse lighting
    - Specular lighting
    - Pyramid
    - Nine Cubes
2. Меню має забезпечувати вибір потрібного режиму роботи
3. У режимі **Diffuse lighting** має показуватися чотирикутник і джерело світла над ним. Джерело світла повинно мати яскравий колір. Потрібно запрограмувати (*ambient + diffuse)* компоненти моделі відбиття світла з урахуванням зменшення освітлення відповідно квадрату відстані до джерела світла.
4. У режимі **Specular lighting** має показуватися чотирикутник і джерело світла над ним. Джерело світла повинно мати яскравий колір. Потрібно запрограмувати (*ambient + specular)* компоненти моделі відбиття світла з постійним освітленням для будь-якої відстані.

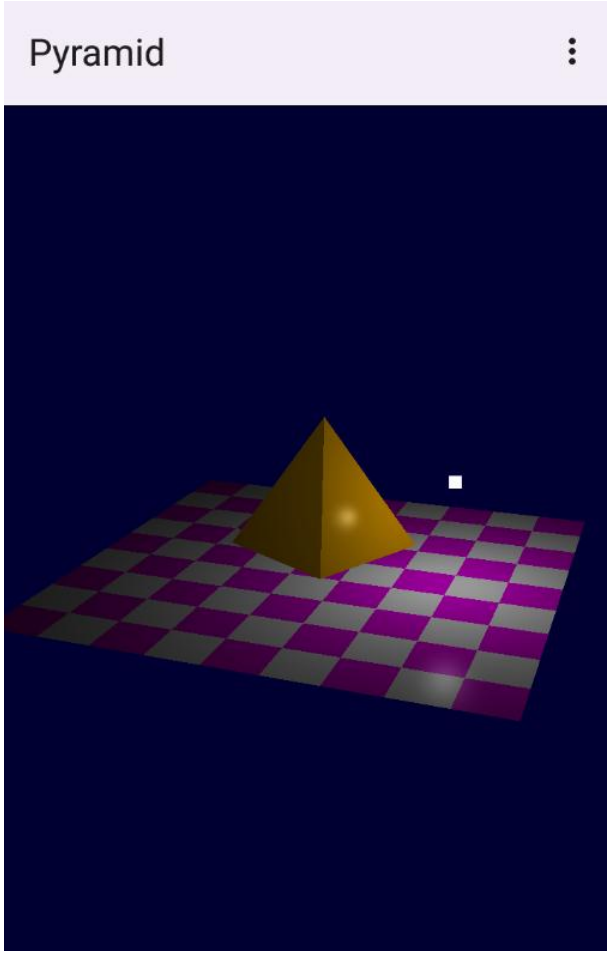




5. У режимі **Pyramid** має показуватися піраміда, яка безперервно обертається над шаховим полем відносно вертикальної осі – режим анімації RENDERMODE_CONTINUOUSLY. Потрібно запрограмувати три (*ambient+diffuse + specular)* компоненти моделі відбиття світла з урахуванням зменшення освітлення відповідно квадрату відстані до джерела світла.

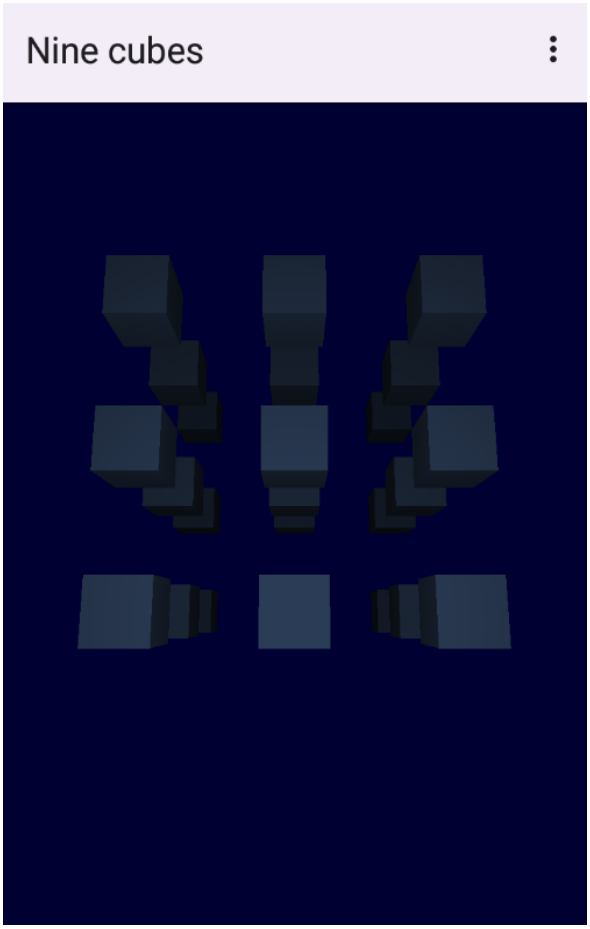
Вибрати розташування джерела щоб наочно продемонструвати дзеркальні бліки відбиття променів від поверхні рухомої піраміди.

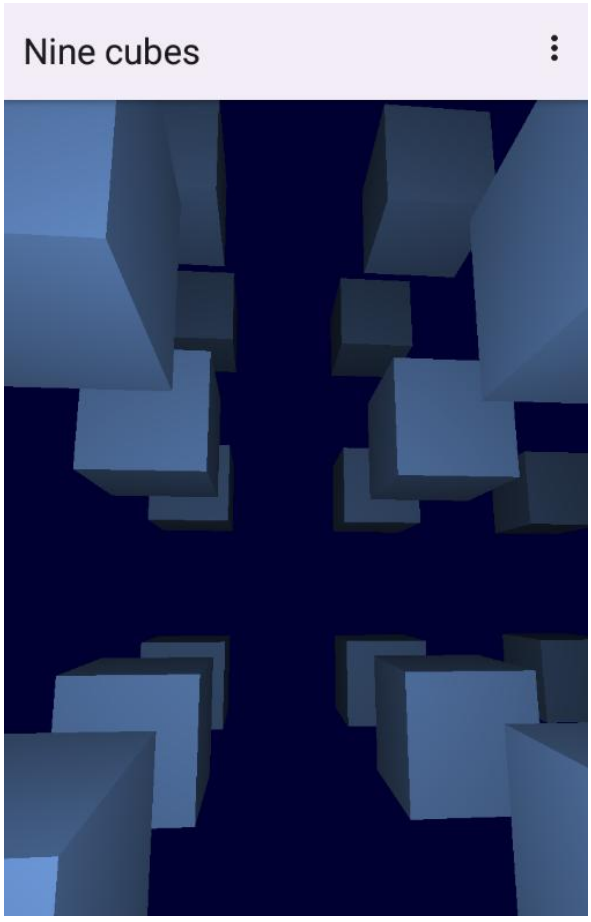




6. У режимах **Diffuse lighting**, **Specular lighting** та **Pyramid** передбачити керування розташуванням джерела світла натискуванням та рухом сенсорів з відповідним оперативним відображенням точкового джерела світла. Для цього використати обробник подій ACTION_DOWN та ACTION_MOVE.

7. У режимі **Nine Cubes** має показуватися решітка з 27 кубів – подібно до попередньої лаб. №3, але вже без шахового поля.чотирикутник і джерело світла над ним. Джерело світла повинно мати яскравий колір. Потрібно запрограмувати (*ambient + diffuse)* компоненти моделі відбиття світла з урахуванням зменшення освітлення відповідно квадрату відстані до джерела світла.





Знайдіть такі коефіцієнти для компонент *ambient* та *diffuse,* щоб при наближенні куби ставали яскравими, а при віддаленні – уходили у темряву.
7. Запрограмувати, щоб у режимі Nine Cubes можна було б за допомогою пересування стілуса (пальця) по екрану змінювати ракурс показу сцени наступним чином (імітувати рух на літальному апараті):

- рухатися вперед-назад вздовж напрямку зору камери
- робити повороти вправо-вліво,
- змінювати нахил камери уверх-вниз і потім відповідно рухатися вздовж нового напрямку зору камери

У цьому режимі точкове джерело світла має бути розташоване у одній точці з камерою. Джерело світла окремо не показувати.

Зміст звіту

1. Титульний аркуш
2. Варіант завдання
3. Вихідний тексти
4. Ілюстрації (скріншоти)
5. Висновки

Контрольні запитання

1. Що таке модель відбиття світла?
2. Як обчислюється колір об'єкта при дифузному відбитті світла?
3. Чим визначається колір об'єкта при дзеркальному відбитті світла?
4. Які компоненти у моделі відбиття світла?
5. Як запрограмувати обчислення косинуса кута між двома векторами?
6. Як знайти координати вектора напрямку променя дзеркального відбиття?
7. Як обчислити координати вектора нормалі?