

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

«Комп'ютерна графіка. Частина 2»

з курсу «Комп'ютерна графіка»

для студентів спеціальностей

151 «Автоматизація та комп'ютерно-інтегровані технології»

172 «Телекомунікація та радіотехніка»

Рекомендовано
редакційно-видавничою
радою університету,
протокол №1 від 19.02.2020р

Харків НТУ «ХПІ» 2020

Методичні вказівки до виконання лабораторних робіт «Комп'ютерна графіка. Частина 2» з курсу «Комп'ютерна графіка» для студентів спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології», 172 «Телекомунікація та радіотехніка» денної та заочної форм навчання / уклад. А. О. Зуєв, О. М. Євсеєнко, В.А. Крилова. – Харків: НТУ«ХП». – 43 с.

Укладачі А. О. Зуєв
О. М. Євсеєнко
В.А. Крилова

Рецензент І.В. Григоренко

Кафедра інформаційно-вимірювальних технологій і систем

ЗМІСТ

Вступ	4
Лабораторна робота № 5. Створення об'єктів сцени	5
Лабораторна робота № 6. Створення мікропрограми для розрахунку освітлення за методом Ламберта для спрямованого джерела світла	12
Лабораторна робота № 7. Створення мікропрограми для розрахунку освітлення за методом Фонга для спрямованого джерела світла	19
Лабораторна робота № 8. Анімація об'єктів сцени	23
Лабораторна робота № 9. Освітлення об'єктів сцени точковим джерелом світла	28
Лабораторна робота № 10. Створення моделі комплексної геометричної фігури і додавання її в сцену	33
Додаток А. Об'єкти у сцені	36
Додаток Б. Параметри джерел світла	37
Додаток В. Параметри відблисків	39
Додаток Г. Швидкості обертання	40
Додаток Д. Швидкість, розмах зсуву джерела по осі Y, радіус освітлення	41
Список літератури	42

ВСТУП

Подання даних на моніторі комп'ютера в графічному вигляді вперше було реалізовано в середині 50-х років для великих ЕОМ, які застосовувалися в наукових і військових дослідженнях. Відтоді графічний спосіб відображення даних став невід'ємною частиною комп'ютерних систем.

Комп'ютерна графіка – це розділ інформатики, який вивчає методи створення та обробки графічних зображень за допомогою обчислювальної техніки.

Комп'ютерна графіка використовується майже в усіх наукових та інженерних дисциплінах для збільшення наочності та поліпшення сприйняття інформації людиною. Без комп'ютерної графіки неможливо уявити собі не тільки комп'ютерний, але і звичайний, цілком матеріальний світ. Незважаючи на те, що комп'ютерна графіка є, по суті, інструментом, її структура і методи засновані на передових досягненнях фундаментальних і прикладних наук: математики, фізики, хімії, біології, статистики, програмування тощо. Це дійсно як для програмних, так і для апаратних засобів створення та обробки зображень на комп'ютері.

Тому комп'ютерна графіка як галузь інформатики активно розвивається і багато в чому виступає рушієм усієї комп'ютерної індустрії.

Найважливішою частиною комп'ютерної графіки є візуалізація – загальна назва прийомів подання інформації або явищ у вигляді, зручному для зорового спостереження та аналізу. Візуалізація даних застосовується в різних сферах людської діяльності: у медицині (комп'ютерна томографія), військовій справі (тренажерні комплекси), наукових дослідженнях (візуалізація будови речовини, векторних полів та інших даних), моделюванні тканин та одягу, дослідно-конструкторських розробках, індустрії розваг.

Методичні вказівки призначені для вивчення основ комп'ютерної графіки та її застосування у створенні систем візуалізації. Розглянуто питання кодування кольору, растеризації, подання та візуалізації тривимірних об'єктів, особливості програмно-апаратних систем тривимірної графіки.

Методичні вказівки містять також індивідуальні завдання для виконання лабораторних і практичних робіт, які допоможуть в ефективному освоєнні курсу комп'ютерної графіки.

ЛАБОРАТОРНА РОБОТА № 5 СТВОРЕННЯ ОБ'ЄКТІВ СЦЕНИ

Мета роботи: Створити кілька різнотипних об'єктів і помістити їх у сцену.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 4.

Загальні відомості

На практиці виникає необхідність візуалізувати більше одного об'єкта одночасно. Для цього доцільно організувати їх у вигляді списку. Разом із загальним інтерфейсом `s_figure` це дозволить однаково створювати, обробляти і видаляти об'єкти.

Для організації такого списку необхідно в кожному об'єкті запам'ятовувати покажчик на об'єкт, наступний за ним, а початок списку запам'ятовувати в глобальній змінній. Таким чином, послідовно переходячи від початку до кінця списку, можна візуалізувати всі об'єкти.

Для створення об'єктів використовується функція `AddFigure()`, а функції `RenderFigure()` і `ReleaseScene()` модифікуються таким чином, щоб обробляти всі об'єкти списку. У функції `InitScene()` створюється кілька об'єктів, які поміщаються в список.

Функція `AddFigure` додає об'єкт до списку і пов'язує його з попередньо створеним об'єктом. Для отримання покажчика на наступний об'єкт у списку використовується інтерфейсна функція `s_figure::next()`. Покажчик на перший об'єкт у списку зберігається в змінній `s_figure* pSceneRoot`.

Фігура створюється оператором `new` із зазначенням параметрів, отриманий покажчик передається в функцію `AddFigure`. Назва класів фігур: площина – `s_plane`, куб – `s_cube`, згладжений куб – `c_cube_smooth`.

Хід виконання:

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 4.
3. Внести зміни в файли проекту згідно з вихідним кодом роботи.
4. Додати опис площини і куба зі згладженими кутами до файлу *figures.cpp*.
5. Додати опис функцій до файлу *kg_lab.h*.
6. Налаштувати і запустити програму. На екрані повинні з'явитися об'єкти сцени: три різнокольорових куба у центрі та сіра площина внизу екрана.
7. Модифікувати об'єкти сцени відповідно до варіанту в функції *InitScene ()*, замінивши функції *AddFigure*, які показані в прикладі (див. додаток А).
8. Налаштувати і запустити програму. На екрані повинні з'явитися об'єкти сцени з заданими в завданні параметрами.
9. На екрані має з'явитися 6 об'єктів. Якщо частини об'єктів не видно, необхідно визначити їх і скоригувати координати і/або кут повороту таким чином, щоб фігури з'явилися в межах видимості (на екрані). Пам'ятайте, що площина – фігура одностороння, і якщо вона орієнтована від спостерігача – не буде відображатися на екрані. У цьому випадку необхідно її повернути таким чином, щоб лицьові грані повернулись до спостерігача.
10. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - схематичний малюнок об'єктів сцени і проекції піраміди видимості у вигляді: зверху (проти осі *Y*), праворуч (проти осі *X*) і спереду (уздовж осі *Z*);
 - текст програм (*scene.cpp*, *figures.cpp*, *kg_lab.h*).

Вихідний код роботи

Додати до файлу `kg_lab.h`:

```
...
struct s_figure
{
    s_vector3    obj_scale, obj_rot, obj_pos;
    ...
    s_figure( const s_vector3& pos_, const s_vector3& rot_ = vec0,
              const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
              float level_ = 0.5f, float spower_ = 16.f, bool anim_ = true );
    s_figure*    child;
    s_figure* next() const { return child; }
    void set_child( s_figure* child_ ) { child = child_; }
};
void ReleaseScene();
void InitScene();
void AddFigure( s_figure* pFig );
}; // namespace scene

namespace figures
{
    using namespace directx;
    struct s_cube: public scene::s_figure
    {
        ...
    };
    struct s_plane: public scene::s_figure
    {
        virtual const s_vertex* verts( DWORD& cnt ) const;
        virtual const t_index* inds( DWORD& cnt ) const;
        s_plane( const s_vector3& pos_, const s_vector3& rot_ = vec0,
                const s_vector3& scale_ = vec1,
                DWORD color_ = 0xff808080UL, float level_ = 0.5f, float spower_ = 16.f ):
            scene::s_figure( pos_, rot_, scale_, color_, level_, spower_, false ) {}
    };
    struct s_cube_smooth: public scene::s_figure
    {
        virtual const s_vertex* verts( DWORD& cnt ) const;
        virtual const t_index* inds( DWORD& cnt ) const;
        s_cube_smooth( const s_vector3& pos_, const s_vector3& rot_ = vec0,
                      const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
                      float level_ = 0.5f, float spower_ = 16.f ):
            scene::s_figure( pos_, rot_, scale_, color_, level_, spower_ ) {}
    };
}; // namespace figures
```

Додати і змінити у файлі scene.cpp:

```
void RenderFigures( IDirect3DDevice9* Dev, LPD3DXEFFECT pEffect )
{
    const s_figure* f = pSceneRoot;
    while( f )
    {
        DWORD vcnt = 0, icnt = 0;
        const s_vertex* v = f->verts( vcnt );
        const t_index* i = f->inds( icnt );
        DWORD tri_cnt = icnt / 3;

        const s_matrix& MatWorld = f->matrix();
        const s_vector4& vObjColor = f->color();

        pEffect->SetMatrix( hMatWorld, &MatWorld );
        pEffect->SetVector(hObjColor, &vObjColor );
        pEffect->SetFloat( hPower, f->power() );
        pEffect->SetFloat( hSpecLevel, f->spec_level() );

        pEffect->CommitChanges();

        Dev->DrawIndexedPrimitiveUP( D3DPT_TRIANGLELIST, 0, vcnt, tri_cnt, i,
            D3DFMT_INDEX16, v, sizeof( s_vertex ) );
        f = f->next();
    } // while( f )
}

s_figure::s_figure( const s_vector3& pos_, const s_vector3& rot_,
    const s_vector3& scale_, DWORD color_, float level_, float spower_,
    bool anim_):
    obj_scale( scale_ ), obj_rot( rot_ ), obj_pos( pos_ ),
    level( level_ ), spower( spower_ ), changes( true ), anim( anim_ ), child( 0 )
{
    D3DXMatrixIdentity( &world );
    set_color( color_ );
}

...

void AddFigure( s_figure* pFig )
{
    pFig->set_child( pSceneRoot );
    pSceneRoot = pFig;
}

void ReleaseScene()
{
    s_figure* f = pSceneRoot;
    while( f )
    {
        s_figure* f1 = f;
        f = f->next();
    }
}
```



```

        delete f1;
    } // while( f )

    pSceneRoot = 0;
}

void InitScene()
{
    vEyePos = vec0;
    vAtPos = s_vector3( 0.f, 0.f, 1.f );
    fFovY = 60.f;
    AddFigure( new figures::s_plane( s_vector3( 0.f, -8.f, 20.f ), s_vector3( 0.f, 0.f, 0.f ),
        s_vector3( 15.f, 1.f, 35.f ), 0xff808080UL, 0.2f, 64.f ) );
    AddFigure( new figures::s_cube( s_vector3( 10.f, -4.f, 30.f ), s_vector3( 0.f, 0.f, 0.f ),
        s_vector3( 3.f, 3.f, 3.f ), 0xff0000ffUL, 1.f, 32.f ) );
    AddFigure( new figures::s_cube_smooth( s_vector3( 0.f, 5.f, 35.f ),
        s_vector3( 0.f,0.f,0.f ),
        s_vector3( 2.f, 2.f, 2.f ), 0xff00ff00UL, 0.5f, 16.f ) );
    AddFigure( new figures::s_cube_smooth( s_vector3( -10.f, -2.f, 25.f ),
        s_vector3( 0.f, 0.f, 0.f ), s_vector3( 3.f, 2.f, 1.f ), 0xffff0000UL, 0.5f, 3.f ) );
}

```

Додати до файлу figures.cpp:

```

...
const t_index* s_cube::inds( DWORD& cnt ) const
{
    ...
    21, 20, 23, 20, 22, 23, //права грань
};
cnt = sizeof( inds ) / sizeof( inds[0] );
return inds;
}
const s_vertex* s_plane::verts( DWORD& cnt ) const
{
    static s_vertex verts[] =
    {
        s_vertex( s_vector3( -1.f, 0.f, -1.f ) ),
        s_vertex( s_vector3( 1.f, 0.f, -1.f ) ),
        s_vertex( s_vector3( -1.f, 0.f, 1.f ) ),
        s_vertex( s_vector3( 1.f, 0.f, 1.f ) ),
    };
    cnt = sizeof( verts ) / sizeof( verts[0] );
    return verts;
}
const t_index* s_plane::inds( DWORD& cnt ) const
{
    static const t_index inds[] =
    {
        1, 0, 2, 1, 2, 3,
    }
}

```

```

};
cnt = sizeof( inds ) / sizeof( inds[0] );
return inds;
}
const s_vertex* s_cube_smooth::verts( DWORD& cnt ) const
{
static s_vertex verts[] =
{
    s_vertex( s_vector3( -1.f, -1.f, -1.f ) ),
    s_vertex( s_vector3( 1.f, -1.f, -1.f ) ),
    s_vertex( s_vector3( -1.f, 1.f, -1.f ) ),
    s_vertex( s_vector3( 1.f, 1.f, -1.f ) ),
    s_vertex( s_vector3( -1.f, -1.f, 1.f ) ),
    s_vertex( s_vector3( 1.f, -1.f, 1.f ) ),
    s_vertex( s_vector3( -1.f, 1.f, 1.f ) ),
    s_vertex( s_vector3( 1.f, 1.f, 1.f ) ),
};
cnt = sizeof( verts ) / sizeof( verts[0] );
return verts;
}
const t_index* s_cube_smooth::inds( DWORD& cnt ) const
{
static const t_index inds[] =
{
    1, 0, 2, 1, 2, 3,      //далека грань
    4, 5, 6, 5, 7, 6,      //ближня грань
    0, 1, 4, 1, 5, 4,      //верхня межа
    6, 7, 2, 7, 3, 2,      //нижня межа
    0, 4, 2, 4, 6, 2,      //ліва грань
    5, 1, 3, 5, 3, 7,      //права грань
};
cnt = sizeof( inds ) / sizeof( inds[0] );
return inds;
}
}; // namespace figures

```

Необхідно додати фігури відповідно до варіанту з таблиці А.2 (див. додаток А), параметри фігур взяті з таблиці А.1 за номером. Назва класів фігур: площина – *s_plane*, куб – *s_cube*, згладжений куб – *c_cube_smooth*.

Контрольні питання

1. Переваги організації візуалізованих об'єктів у вигляді списку.
2. Що необхідно зробити для організації списку?
3. Призначення функцій *AddFigure()*, *RenderFigures()* і *ReleaseScene()*.
4. Призначення функції *InitScene()*.
5. Яка функція використовується для отримання покажчика на наступний об'єкт у списку?
6. В якій змінній зберігається покажчик на перший об'єкт у списку?
7. Що таке *s_plane*, *s_cube*, *c_cube_smooth*?

ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ МІКРОПРОГРАМИ ДЛЯ РОЗРАХУНКУ ОСВІТЛЕННЯ ЗА МЕТОДОМ ЛАМБЕРТА ДЛЯ СПРЯМОВАНОГО ДЖЕРЕЛА СВІТЛА

Мета роботи: Додати опис спрямованого джерела світла, створити мікропрограму, яка розраховує освітлення сцени джерелом світла із заданими параметрами.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 5.

Загальні відомості

Для підвищення реалістичності синтезованих зображень необхідно розраховувати освітленість кожного елемента тривимірної сцени що візуалізується. Такий розрахунок доцільно проводити, використовуючи обчислювальні потужності прискорювача тривимірної графіки. Для цього необхідні обчислення треба помістити у фрагментну мікропрограму (файл *shader.fx*).

Для освітлення сцени використовуємо два джерела:

1) джерело фонового освітлення (*ambient*), яке характеризується кольором випромінюваного світла C_a і рівномірно освітлює об'єкти з усіх напрямків;

2) джерело спрямованого освітлення (*directional*), яке освітлює об'єкти із заданого напрямку L , колір джерела C_d .

Для розрахунку освітленості використовуємо правило Ламберта: інтенсивність освітлення i залежить від кута a між нормаллю до поверхні N і напрямком падаючого світла за законом косинуса $i = \cos a$.

Для визначення величини $\cos a$ необхідно нормалізувати вектори L і N і розрахувати скалярний добуток між ними (замість вектора L використовується вектор L' , який має протилежний зміст): $i = \max(0, |L'| \cdot |N|)$. Оскільки інтенсивність освітлення не може приймати негативні значення, потрібно взяти максимум з 0 і величини отриманого скалярного добутку. Отриманий вираз реалізується у функції *lambert_law()*.

Сумарна освітленість i_s двома джерелами розраховується таким чином: $i_s = i * C_d + C_a$. Дана величина множить на колір об'єкта і повертається в результаті фрагментною мікропрограмою *ps_lambert()*.

Для правильного розрахунку освітленості необхідно розрахувати для кожної вершини фігур нормаль. Цю операцію здійснює функція *ComputeNormals()*, яка викликається в методі *comp_normals()* інтерфейсу *s_figure* під час конструювання фігур.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 5.
3. Внести зміни до файлів проекту згідно з вихідним кодом роботи.
4. Налаштувати і запустити програму. На екрані повинні з'явитися об'єкти сцени відповідно до варіанту з лабораторної роботи № 5.
5. Задати параметри джерела світла в функції *InitScene()* відповідно до номера варіанта. Замість знаків **AAAA**, **LLLL**, **DDD** повинні бути задані коди кольорів фонового і спрямованого джерел світла, напрямком на джерело світла (див. додаток Б).
6. Налаштувати і запустити програму. На екрані повинні з'явитися об'єкти сцени, освітлені джерелом світла із заданими в завданні параметрами.
7. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - схематичний малюнок об'єктів сцени з вказівкою напрямку світла, випромінюваного джерелом;
 - значення нормалізованого вектора **DDD**;
 - текст програм (*scene.cpp*, *shader.fx*).

Вихідний код роботи

Додати до файла `kg_lab.h`:

```
...
namespace render
{
enum
{
    RM_WIRE,
    RM_LAMBERT,
    RM_MAX,
};
```

змінити рядок `const int CurRenderMethod = ...` на
`const int CurRenderMethod = RM_LAMBERT;`

```
...
```

Додати метод `comp_normals()` у структуру `s_figure` (у самий кінець, перед дужкою, яка закривається):

```
struct s_figure
{
    ...
    void comp_normals();
};
```

Додати виклик методу `comp_normals()` у конструктор кожної фігури для автоматичного розрахунку нормалей:

```
namespace figures
{
using namespace directx;
struct s_cube: public scene::s_figure
{
    ...
    scene::s_figure( pos_, rot_, scale_, color_, level_, spower_ ) { comp_normals(); }
};
struct s_plane: public scene::s_figure
{
    ...
    scene::s_figure( pos_, rot_, scale_, color_, level_, spower_, false )
    { comp_normals(); }
};
struct s_cube_smooth: public scene::s_figure
{
    ...
    scene::s_figure( pos_, rot_, scale_, color_, level_, spower_ )
    { comp_normals(); }
};
}; // namespace figures
```

Додати до файлу render.cpp:

```
bool LoadResources( IDirect3DDevice9* Dev )
{
...
if( !pEffect )
{
    main::message_box_( L"Шейдер не знайдено!" );
    return false;
} // if( !pEffect )
static const char* tnames [RM_MAX] = { "wire", "lambert" };
for( int i = 0; i < RM_MAX; ++i )
...

```

Додати до файлу scene.cpp:

```
namespace scene
{
s_figure* pSceneRoot = 0;
s_vector4    vLigColor;
s_vector4    vAmbColor;
s_vector4    vLigDir;
s_vector4    vLigPos;
...
void color2vec( DWORD color_, s_vector4& obj_color )
{
...
}

void ComputeNormals( s_vertex* verts, int vcnt, const t_index* inds, int icnt )
{
for( int v = 0; v < vcnt; ++v )
{
    verts[v].Norm = vec0;
} // for( int v = 0; v < vcnt; ++v )
int tris = icnt / 3;
for( int t = 0; t < tris; ++t )
{
    int i = t * 3;
    const D3DXVECTOR3& v0 = verts[inds[i + 0]].Pos;
    const D3DXVECTOR3& v1 = verts[inds[i + 1]].Pos;
    const D3DXVECTOR3& v2 = verts[inds[i + 2]].Pos;

    s_vector3 ledge, redge;
    D3DXVec3Subtract( &ledge, &v0, &v1 );
    D3DXVec3Subtract( &redge, &v0, &v2 );

    s_vector3 norm, nnorm;
    D3DXVec3Cross( &norm, &ledge, &redge );

```

```

D3DXVec3Normalize( &norm, &norm );

for( int v = 0; v < 3; ++v )
{
    s_vertex& vert = verts[inds[i + v]];
    D3DXVec3Add( &vert.Norm, &vert.Norm, &norm );
} // for( int v = 0; v < 3; ++v )

} // for( int t = 0; t < tris; ++t )

for (int v = 0; v < vcnt; ++v )
{
    D3DXVec3Normalize( &verts[v].Norm, &verts[v].Norm );
} // for( int v = 0; v < vcnt; ++v )
}

void RenderFigures( IDirect3DDevice9* Dev, LPD3DXEFFECT pEffect )
{
    pEffect->SetVector( hLigColor, &vLigColor );
    pEffect->SetVector( hAmbColor, &vAmbColor );
    pEffect->SetVector( hLigDir, &vLigDir );
    pEffect->SetVector( hLigPos, &vLigPos );
    ...
    s_figure::s_figure()
    {
        ...
    }

    void s_figure::comp_normals()
    {
        DWORD vcnt = 0, icnt = 0;
        s_vertex* v = (s_vertex*) verts( vcnt );
        const t_index* i = inds( icnt );
        ComputeNormals( v, vcnt, i, icnt );
    }

    const s_matrix& s_figure::matrix() const
    {
        ...
    }

    void InitScene()
    {
        vEyePos = vec0;
        vAtPos = s_vector3( 0.f, 0.f, 1.f );
        fFovY = 60.f;

        color2vec( LLLL, vLigColor );
        color2vec( AAAA, vAmbColor );
        vLigDir = s_vector4( DDD, 0.f );
        vLigPos = s_vector4( 0.f, 0.f, 20.f, 35.f );
    }

```



```
AddFigure ( new figures::
```

```
...  
}
```

Увага! У функції *InitScene()* замість знаків **AAAA**, **LLLL**, **DDD** повинні бути задані коди кольорів фонового і спрямованого джерел світла, напрямком на джерело світла (див. додаток Б).

Додати до файлу **shader.fx**:

```
...
```

```
technique wire
```

```
{  
    pass P0  
    {
```

```
...
```

```
    }  
}
```

```
float3 lambert_law( float3 L, float3 N )
```

```
{  
    float LdN = max( 0.0, dot( L, N ) );  
    return v_lig_color.rgb * LdN;  
}
```

```
float4 ps_lambert( s_output2 i ): COLOR
```

```
{  
    float3 color = v_obj_color.rgb;
```

```
    float3 N = normalize( i.nor );  
    float3 L = normalize( v_lig_dir.xyz );
```

```
    float3 ambient = v_amb_color.rgb;  
    float3 diffuse = lambert_law( L, N );
```

```
    return float4( ( ambient + diffuse ) * color, 1.f );  
}
```

```
technique lambert
```

```
{  
    pass P0  
    {
```

```
        FillMode           = Solid;  
        FogEnable          = false;  
        AlphaBlendEnable   = false;  
        AlphaTestEnable    = false;  
        ZEnable            = true;  
        ZFunc               = LessEqual;  
        CullMode           = CCW;
```

```
        VertexShader       = compile vs_2_0 vs_simple();
```

```
        PixelShader          = compile ps_2_0 ps_lambert();
    }
}
```

Увага! Перед заданням вектора **DDD** його необхідно нормалізувати: розрахувати його довжину і кожну компоненту вектора розділити на неї.

Контрольні питання

1. Навіщо потрібна освітленість синтезованих зображень?
2. Які джерела використовуються для освітлення сцени?
3. Чим характеризується джерело фонового освітлення?
4. Чим характеризується джерело спрямованого освітлення?
5. Правило Ламберта.
6. Як розрахувати \cos кута α між двома векторами.
7. Формула для розрахунку сумарної освітленості.
8. Призначення функції *ComputeNormals()*.

ЛАБОРАТОРНА РОБОТА № 7 СТВОРЕННЯ МІКРОПРОГРАМИ ДЛЯ РОЗРАХУНКУ ОСВІТЛЕННЯ ЗА МЕТОДОМ ФОНГА ДЛЯ СПРЯМОВАНОГО ДЖЕРЕЛА СВІТЛА

Мета роботи: Вивчити принцип розрахунку освітлення сцени за методом Фонга.

Необхідне обладнання та комплектуючі, ПЗ: лабораторні роботи № 5 і № 6.

Загальні відомості

Для підвищення реалістичності синтезованих зображень крім розсіяного освітлення необхідно розраховувати відблискову складову освітлення. Такий розрахунок доцільно проводити, використовуючи обчислювальні потужності прискорювача тривимірної графіки. Для цього потрібні обчислення необхідно помістити в фрагментну мікропрограму (файл *shader.fx*).

Для розрахунку відблисків скористаємося методом Фонга, для цього додатково додамо два параметри для спрямованого джерела світла: інтенсивність відблиску C_s і розмір відблиску p .

Інтенсивність відблиску згідно з методом Фонга: $i_\phi = C_s * (V.R)^p$. Для проведення розрахунків необхідно обчислити вектор відбитого світла: $R = L' - 2 * N * (L'.N)$, де N – нормаль до поверхні в точці, для якої ведеться розрахунок освітлення, а L' – напрямок на джерело світла, і вектор напрямлення на спостерігача $V = |E - P|$, де E – позиція камери, а P – координати точки, для якої ведеться розрахунок.

Для розрахунку вектора відображення R використовується функція *reflect()*.

Отриману інтенсивність i_ϕ необхідно модулювати величиною $L.N$, для того щоб відблиски не виникали на затіненій стороні об'єктів. У мікропрограмі ця модуляція здійснюється шляхом множення інтенсивності на величину $\max(1, L.N * 10)$.

Сумарна освітленість i_s' (з урахуванням відблисків) обчислюється так: $i_s' = C_d * (i + i_\phi) + C_a$. Розрахунок інтенсивності відблиску ведеться у фрагментній мікропрограмі *ps_phong()*.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 6.
3. Внести зміни у файли проекту згідно з вихідним кодом роботи.
4. Налагодити і запустити програму. На екрані повинні з'явитися об'єкти сцени відповідно до варіанту з лабораторної роботи № 6, освітлені джерелами світла згідно з додатком Б лабораторної роботи № 6.
5. Задати параметри відблисків для кожного з об'єктів у функції *InitScene ()* відповідно до номера варіанта. Замість знаків **SSS**, **PPP** повинні бути задані: інтенсивність відблиску і розмір відблиску (див. додаток В).
6. Налагодити і запустити програму. На екрані повинні з'явитися об'єкти сцени, освітлені джерелом світла із заданими в завданні параметрами.
7. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - текст програм (*scene.cpp*, *shader.fx*).

Вихідний код роботи

Додати у файл `kg_lab.h`:

```
...
namespace render
{
enum
{
    RM_WIRE,
    RM_LAMBERT,
    RM_PHONG,
    RM_MAX,
};
const int CurRenderMethod = RM_PHONG;
const DWORD BackColor = 0xff000000UL;
bool LoadResources( IDirect3DDevice9* Dev );
void RenderScene( IDirect3DDevice9* Dev, int TN );

```

Додати до файлу `scene.cpp` для фігур, створюваних функцією `AddFigure()` відповідно до варіанту:

приклад:

```
AddFigure( new figures::s_plane( s_vector3( 0.f, -8.f, 20.f ), s_vector3( 0.f, 0.f, 0.f ),
s_vector3( 15.f, 1.f, 35.f ), 0xff808080UL, SSS, PPP ) );
```

Замість параметрів SSS і PPP підставити для кожного об'єкта інтенсивність і розмір відблиску згідно з варіантом (див. додаток В). Інші параметри залишити незмінними.

Додати в кінець файлу `shader.fx`:

```
...
float3 phong_law( float3 L, float3 N, float3 V )
{
    float LdN = max( 0.0, dot( L, N ) );
    float3 R = reflect( -L, N );
    float3 specular = v_lig_color.rgb * f_spec_level *
        pow( max( 0.0, dot( V, R ) ), f_power );
    specular *= min( 1.0, LdN * 10.0 );
    return specular;
}
float4 ps_phong( s_output2 i ): COLOR
{
    float3 color = v_obj_color.rgb;
    float3 N = normalize( i.nor );
    float3 V = -normalize( i.eye );
    float3 L = normalize( v_lig_dir.xyz );
    float3 ambient = v_amb_color.rgb;
    float3 diffuse = lambert_law( L, N );
    float3 specular = phong_law( L, N, V );

```

```

        return float4( (ambient + diffuse) * color + specular, 1.f);
    }
    technique phong
    {
        pass P0
        {
            FillMode           = Solid;
            FogEnable           = false;
            AlphaBlendEnable   = false;
            AlphaTestEnable    = false;
            ZEnable             = true;
            ZFunc               = LessEqual;
            CullMode            = CCW;
            VertexShader        = compile vs_2_0 vs_simple();
            PixelShader         = compile ps_2_0 ps_phong();
        }
    }
}

```

Додати до файлу **render.cpp**:

```

bool LoadResources( IDirect3DDevice9* Dev )
{
    ...
    if( !pEffect )
    {
        main::message_box_( L"Шейдер не знайдено!" );
        return false;
    } // if( !pEffect )
    static const char* tnames[RM_MAX] = { "wire", "lambert", "phong" };
    for( int i = 0; i < RM_MAX; ++i )
    {
        ...
    }
}

```

Контрольні питання

1. Навіщо необхідно використовувати відблискову складову освітлення?
2. Метод Фонга.
3. Формула для розрахунку інтенсивності відблиску за методом Фонга.
4. Якою функцією необхідно скористатися для розрахунку вектора відображення?
5. Як розраховується сумарна освітленість?

ЛАБОРАТОРНА РОБОТА № 8 АНІМАЦІЯ ОБ'ЄКТІВ СЦЕНИ

Мета роботи: Дослідити принципи анімації об'єктів сцени з урахуванням часу синтезу кадру.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 7.

Загальні відомості

Для додатків, запущених під ОС Windows, а також інших багатозадачних ОС важко домогтися постійного часу синтезу кадру для послідовних кадрів. При запуску процесів ОС, інших програм час, що виділяється додатку, може варіюватися в широких межах. Так само конфігурації ПК, на яких запускається додаток, можуть бути різними. Це призводить до того, що гарантовано, заздалегідь (під час компіляції програми) неможливо задати точні часові інтервали.

Для створення плавної анімації об'єктів у сцені (обертання або зміщення) необхідно враховувати час, який минув з моменту початку синтезу попереднього кадру. Для отримання часу в Win32 API існує ряд функцій. У лабораторній роботі скористаємося найпростішою *GetTickCount()* – ця функція дозволяє отримати час, що минув з моменту запуску ОС, у мілісекундах.

Для того щоб розрахувати час, що минув з моменту початку синтезу попереднього кадру до початку синтезу поточного, необхідно запам'ятати час t_1 , який поверне функція *GetTickCount()*, а потім, перед початком синтезу кадру розрахувати різницю між поточним часом t_2 , який поверне *GetTickCount()* і t_1 . Отримане значення $\Delta T = t_2 - t_1$ після масштабування (перетворення в секунди) можна використовувати для масштабування кроку анімації.

У лабораторній роботі функція *PeekTimeDelta()* повертає величину ΔT в секундах. У функції *UpdateSceneObjects()* відбувається обчислення кроку анімації з урахуванням масштабування. Далі всі об'єкти, які знаходяться в сцені і для яких встановлено прапор анімації, повертаються на знову обчислені кути.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 7.
3. Додати до проекту файл *animation.cpp*.
4. Внести зміни до файлів проекту згідно з вихідним кодом роботи.
5. Задати параметри швидкості обертання по осях (замість знаків RX, RY) В функції *UpdateSceneObjects()* відповідно до номера варіанта (див. додаток Г).
6. Налагодити і запустити програму. На екрані повинні з'явитися об'єкти сцени, що обертаються, відповідно до варіанту з лабораторної роботи № 6.
7. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - текст програм (*animation.cpp*).

Вихідний код роботи

Додати до файлу `kg_lab.h`:

```
...
struct s_figure
{
...
    bool is_anim() const { return anim; }
};
...
namespace scene
{
...
    void InitScene();
    void AddFigure( s_figure* pFig );
    void UpdateScene();

}; // namespace scene

namespace figures
{
using namespace directx;
struct s_cube: public scene::s_figure
{
    virtual const s_vertex* verts( DWORD& cnt ) const;
    virtual const t_index* inds( DWORD& cnt ) const;
    s_cube( const s_vector3& pos_, const s_vector3& rot_ = vec0,
           const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
           float level_ = 0.5f, float spower_ = 16.f, bool anim_ = true ):
        scene::s_figure( pos_, rot_, scale_, color_, level_, spower_, anim_ )
        { comp_normals(); }
};
...
struct s_cube_smooth: public scene::s_figure
{
    virtual const s_vertex* verts( DWORD& cnt ) const;
    virtual const t_index* inds( DWORD& cnt ) const;
    s_cube_smooth( const s_vector3& pos_, const s_vector3& rot_ = vec0,
                  const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
                  float level_ = 0.5f, float spower_ = 16.f, bool anim_ = true ):
        scene::s_figure( pos_, rot_, scale_, color_, level_, spower_, anim_ )
        { comp_normals(); }
};
...
}; // namespace figures
```

```

namespace animation
{
    using namespace directx;
    float PeekTimeDelta();
    void UpdateSceneObjects( scene::s_figure* pRoot );
}; // namespace animation

```

Додати до файлу scene.cpp:

```

void UpdateSceneMatrices( LPD3DXEFFECT pEffect )
{
    ...
}
void UpdateScene ()
{
    animation::UpdateSceneObjects( pSceneRoot );
}
}; // namespace scene

```

Додати до файлу directx.cpp:

```

void Pump()
{
    if( !main::boActive )
        Sleep( 20 );
    scene::UpdateScene();
    Paint();
}

```

Додати до файлу animation.cpp:

```

#include "stdafx.h"
#include "kg_lab.h"
namespace animation
{
    float fYaw = 0.f;
    float fPitch = 0.f;
    //отримання часу який пройшов з минулого кадру
    float PeekTimeDelta()
    {
        static int t = GetTickCount();
        int dt = GetTickCount() - t;
        t = GetTickCount();
        return 0.001f * Dt;
    }
    void UpdateSceneObjects( scene::s_figure* pRoot )
    {
        float dt = PeekTimeDelta();
        fYaw += dt * RX;
        fPitch += dt * RY;
        while( pRoot )

```

```
{
    if( pRoot->is_anim() )
        pRoot->rotate( s_vector3( fYaw, fPitch, 0.f ) );
    pRoot = pRoot->next ();
} // while( pRoot )
}
```

Контрольні питання

1. Назвіть причини, через які важко домогтися постійного часу синтезу кадру.
2. Що потрібно враховувати при створенні плавної анімації?
3. Як отримати час в Win32 API?
4. Призначення функції *GetTickCount()*?
5. Призначення функцій *PeekTimeDelta()* і *UpdateSceneObjects()*.

ЛАБОРАТОРНА РОБОТА № 9 ОСВІТЛЕННЯ ОБ'ЄКТІВ СЦЕНИ ТОЧКОВИМ ДЖЕРЕЛОМ СВІТЛА

Мета роботи: Дослідити принципи роботи точкового джерела світла.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 8.

Загальні відомості

Більш складним є моделювання освітлення від точкового джерела світла. Для такого типу джерела додатково необхідно задавати позицію і радіус R освітлюваної джерелом сфери (у лабораторній зберігатися у змінній *vLigPos* – позиція в полях x, y, z , а радіус у полі w).

Об'єкти, що знаходяться на відстані понад R від центра джерела, ним не освітлюються. Чим ближче точка об'єкта знаходиться до джерела світла, тим інтенсивніше вона освітлюється. Для моделювання такого затухання використовується квадратична залежність від відстані. Коефіцієнт затухання a_t розраховується в піксельній мікропрограмі за такою формулою: $a_t = 1 - \min(0, L/R)^2$, де L – відстань від фрагмента до центра джерела світла.

Ще однією відмінністю точкового джерела від спрямованого є те, що напрямок на джерело світла не є константою, а розраховується для кожної точки індивідуально: $L' = L_p - P$, де L_p – позиція джерела світла, а P – координати точки.

Розрахунок освітлення ведеться за правилами Фонга і Ламберта, аналогічно спрямованому джерелу світла. Отримані інтенсивності модулюються (множаться) коефіцієнтом загасання a_t .

У лабораторній роботі анімація джерела світла ведеться шляхом зміни його позиції по вертикалі з певною швидкістю.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 8.
3. Внести зміни до файлів проекту згідно з вихідним кодом роботи.
4. Задати параметри швидкості і розмаху зміщення джерела по осі Y (замість знаків **SSS**, **DDD**) В функції *UpdateSceneObjects()* відповідно до номера варіанта (див. додаток Б).
5. Задати радіус джерела світла (замість знака **RRR**) В функції *InitScene()* відповідно до номера варіанта (див. додаток Д).
6. Налогодити і запустити програму. На екрані повинні з'явитися обертові об'єкти сцени відповідно до варіанта з лабораторної роботи № 6 і анімоване джерело світла, яке переміщується у вертикальній площині.
7. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - текст програм (*animation.cpp*, *shader.fx*).

Додаток А. Вихідний код роботи

Додати до файлу *kg_lab.h*:

```
...
namespace render
{
enum
{
    RM_WIRE,
    RM_LAMBERT,
    RM_PHONG,
    RM_PHONG_POINT,
    RM_MAX,
};
const int CurRenderMethod = RM_PHONG_POINT;
const DWORD BackColor = 0xff000000UL;
bool LoadResources( IDirect3DDevice9* Dev );
...
namespace scene
{
...
void InitScene();
void AddFigure( s_figure* pFig );
void UpdateScene();
}
```

```
extern s_vector4 vLigPos;

}; // namespace scene
```

Додати до файлу **render.cpp**:

```
bool LoadResources( IDirect3DDevice9* Dev )
{
...
if( !pEffect )
{
    main::message_box_( L"Шейдер не знайдено!" );
    return false;
} // if( !pEffect )
static const char* tnames[RM_MAX] =
{ "wire", "lambert", "phong", "phong_point" };

for( int i = 0; i < RM_MAX; ++i )
{
...
}
```

Додати до файлу **animation.cpp**:

```
#include "stdafx.h"
#include "kg_lab.h"
namespace animation
{
float fYaw = 0.f;
float fPitch = 0.f;
float fLightH = 0.f;
...
void UpdateSceneObjects( scene::s_figure* pRoot )
{
...
    pRoot = pRoot->next ();

} // while( pRoot )
fLightH += dt * SSS;
if( fLightH > 1.f )
    fLightH -= (int)fLightH;
scene::vLigPos.y = sinf( fLightH * 3.14f ) * DDD - 7.5f;
}
}; // namespace animation
```

Замінити у файлі scene.cpp:

```
...
void InitScene()
{
vEyePos = vec0;
vAtPos = s_vector3( 0.f, 0.f, 1.f );
fFovY = 60.f;
color2vec( 0xffffffffUL, vLigColor );
color2vec( 0xff404040UL, vAmbColor );
vLigDir = s_vector4( 0.5f, 1.f, 0.5f, 0.f );
vLigPos = s_vector4( 0.f, 0.f, 20.f, RRR );

```

...

Додати в кінець файлу shader.fx:

```
...
float4 ps_phong_point( s_output2 i ): COLOR
{
float3 color = v_obj_color.rgb;
float3 N = normalize( i.nor );
float3 V = -normalize( i.eye );
float3 D = v_lig_pos.xyz - i.wpos;
float3 L = normalize( D );
float3 ambient = v_amb_color.rgb;
float3 diffuse = lambert_law( L, N );
float3 specular = phong_law( L, N, V );
float rad = v_lig_pos.w, ln = length( D );
float att = 1.0 - saturate( ln / rad ), att2 = sqrt( att );
specular * = att;
diffuse * = att;
return float4( ( ambient + diffuse ) * color + specular, 1.f );
}

technique phong_point
{
pass P0
{
FillMode           = Solid;
FogEnable           = false;
AlphaBlendEnable   = false;
AlphaTestEnable    = false;
ZEnable            = true;
ZFunc              = LessEqual;
CullMode           = CCW;
VertexShader       = compile vs_2_0 vs_simple();
PixelShader        = compile ps_2_0 ps_phong_point();
}
}

```

Контрольні питання

1. Що таке точкове джерело світла?
2. Які параметри необхідно ставити для використання точкового джерела світла?
3. Як залежить інтенсивність освітленості об'єкта від відстані до джерела освітлення?
4. За якою формулою розраховується напрямок на джерело світла?
5. Що таке коефіцієнт затухання?
6. Як ведеться розрахунок освітлення за правилами Фонга і Ламберта?

7. ЛАБОРАТОРНА РОБОТА № 10 СТВОРЕННЯ МОДЕЛІ КОМПЛЕКСНОЇ ГЕОМЕТРИЧНОЇ ФІГУРИ І ДОДАВАННЯ ЇЇ В СЦЕНУ

Мета роботи: Створення моделі складної геометричної фігури.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 9, завдання № 3 з РГЗ.

Загальні відомості

Для опису геометричної фігури в лабораторній роботі використовується загальний інтерфейс *s_figure*, який дозволяє проводити конструювання фігури, зміну її параметрів і отримувати опис геометрії і топології фігури. Опис фігур поміщається в простір імен *figures*. Нова фігура створюється шляхом успадкування інтерфейсу *s_figure*. Наприклад, у такий спосіб: *struct s_my_figure: public scene::s_figure{}*.

Для кожної фігури необхідно визначити дві функції: *verts()* – повертає список вершин фігури і *inds()* – повертає топологію фігури (трійки індексів описують трикутники). У середині кожної з функцій міститься відповідно масив вершин у локальній системі координат і масив індексів.

Для задання вершин використовується структура *s_vertex*, конструктор якої бере 1 обов'язковий аргумент типу *s_vector3* (вектор з 3-х компонент), який містить координати вершини. Об'ява вершини проводиться таким чином: *s_vertex(s_vector3(x, y, z))*, де *x, y, z* – координати вершини. Вершини поміщаються в масив у функції *verts()* і розділяються комами.

Для задання топології фігури використовуються трійки індексів (номерів вершин), що визначають трикутники, з яких складається фігура. Нумерація вершин починається з 0-го елемента.

Номери вершин задаються в масиві в функції *inds()* і відокремлюються один від одного комами. Пам'ятайте, що номер вершини не повинен перевищувати кількість вершин у масиві вершин, заданому в функції *verts()*. Наприклад, для площини номери вершин будуть: 1, 0, 2, 1, 2, 3. При цьому кількість вершин дорівнює чотирьом.

Для задання трикутників важливий порядок обходу вершин. У лабораторній роботі прийнято, що видимі трикутники задаються за годинниковою стрілкою, а невидимі – проти годинникової. Цей порядок встановлюється у файлі *shader.fx* в параметрі *CullMode*. При значенні *CCW*

видаляються трикутники, які мають порядок проти годинникової стрілки, при значенні *CW* – за годинниковою, а при значенні *none* видалення не відбувається.

Хід виконання

1. Запустити середовище розробки Microsoft Visual C++ Express Edition.
2. Відкрити проект лабораторної роботи № 9.
3. Створити структуру опису фігури *s_my_figure* відповідно до варіанту з РГЗ (див. вихідний код роботи).
4. Замінити одну з фігур, які знаходяться в сцені, новоствореною фігурою. Для цього в функції *InitScene()* в одному з викликів функцій *AddFigure()* змінити тип фігури на *s_my_figure*, інші параметри залишити без змін.
5. Налаштувати і запустити програму. На екрані повинні з'явитися фігури, додані в сцену, освітлені точковим джерелом світла.
6. Для налагодження новоствореної фігури можна перемкнути режим візуалізації на каркасну модель, замінивши константу *CurRenderMethod* у файлі *kg_lab.h* на *RM_WIRE*:
const int CurRenderMethod = RM_WIRE.
7. Підготувати звіт про лабораторну роботу, який повинен містити:
 - завдання;
 - фігуру з РГЗ у вигляді зверху і у вигляді праворуч;
 - текст програм (*figures.cpp*).

Вихідний код роботи

Додати до файлу `kg_lab.h` структуру опису нової фігури `s_my_figure`:

```
...
namespace figures
{
...
struct s_my_figure: public scene::s_figure
{
    virtual const s_vertex* verts( DWORD& cnt ) const;
    virtual const t_index* inds( DWORD& cnt ) const;
    s_my_figure( const s_vector3& pos_, const s_vector3& rot_ = vec0,
                const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
                float level_ = 0.5f, float spower_ = 16.f, bool anim_ = true ):
    scene::s_figure( pos_, rot_, scale_, color_, level_, spower_, anim_ )
        { comp_normals (); }
};
}; // namespace figures
...
```

Додати в кінець файла `figures.cpp` опис вершин і топології (трикутників) нової фігури `s_my_figure` згідно з завданням № 3 з РГЗ:

```
...
const s_vertex* s_my_figure::verts( DWORD& cnt ) const
{
    static s_vertex verts[] =
    {
        //у цьому місці розмістити координати вершин фігури
        //s_vertex( s_vector3( Xf, Yf, Zf ) ), де X, Y, Z – координати вершин.
        ...
    };
    cnt = sizeof( verts ) / sizeof( verts[0] );
    return verts;
}
const t_index* s_my_figure::inds( DWORD& cnt ) const
{
    static const t_index inds[] =
    {
        //у цьому місці розмістити трійки індексів,
        //які описують топологію фігури (розділяючи їх комами)
        ...
    };
    cnt = sizeof( inds ) / sizeof( inds[0] );
    return inds;
}
}; // namespace figures
```

ДОДАТКИ
Додаток А
Об'єкти у сцені

Таблиця А.1

№ Фігури	Колір (####)	Масштаб (SSS)	Обертання (RRR)	Зсув (TTT)
1	червоний	2.f, 2.f, 2.f	45.f, 0.f, 0.f	10.f, -4.f, 30.f
2	зелений	3.f, 3.f, 3.f	30.f, 30.f, 0.f	-4.f, -4.f, 25.f
3	білий	1.f, 1.f, 1.f	0.f, 0.f, 0.f	1.f, 0.f, 10.f
4	сірий	2.f, 3.f, 3.f	45.f, 0.f, 0.f	-5.f, 8.f, 25.f
5	жовтий	3.f, 3.f, 3.f	30.f, 30.f, 0.f	-4.f, -4.f, 25.f
6	ціан	2.f, 3.f, 3.f	0.f, 0.f, 0.f	1.f, 0.f, 25.f
7	пурпурний	2.f, 2.f, 2.f	0.f, 0.f, 60.f	-1.f, -0.5f, 8.f
8	синій	2.f, 3.f, 3.f	30.f, 30.f, 0.f	-8.f, 0.f, 35.f
9	фіолетовий	3.f, 1.f, 2.f	45.f, 0.f, 0.f	1.f, 0.f, 25.f
10	помаранчевий	5.f, 1.f, 1.f	0.f, 0.f, 60.f	-4.f, -4.f, 25.f
11	зелений	1.f, 1.f, 1.f	0.f, 0.f, 0.f	-5.f, 8.f, 25.f
12	червоний	3.f, 3.f, 3.f	0.f, 0.f, 60.f	-8.f, 0.f, 35.f
13	синій	2.f, 3.f, 3.f	30.f, 30.f, 0.f	10.f, -4.f, 30.f
14	ціан	3.f, 1.f, 2.f	0.f, 0.f, 60.f	-4.f, -4.f, 25.f
15	білий	2.f, 3.f, 3.f	0.f, 0.f, 0.f	-5.f, 8.f, 25.f

Таблиця А.2

№ вар.	Тип фігури і номер параметрів з таблиці 1					
	1	2	3	4	5	6
1	Куб, 3	Куб, 5	Куб, 12	Куб, 1	Площина, 4	Згл. куб, 15
2	Площина, 13	Площина, 7	Згл. куб, 15	Згл. куб, 8	Куб, 4	Куб, 10
3	Згл. куб, 3	Куб, 11	Куб, 5	Площина, 4	Куб, 1	Згл. куб, 2
4	Площина, 11	Куб, 1	Згл. куб, 2	Згл. куб, 3	Площина, 13	Куб, 5
5	Куб, 1	Згл. куб, 15	Куб, 4	Згл. куб, 8	Згл. куб, 2	Площина, 12
6	Згл. куб, 2	Куб, 3	Площина, 12	Куб, 6	Куб, 7	Згл. куб, 5
7	Площина, 11	Куб, 10	Площина, 13	Згл. куб, 3	Куб, 6	Куб, 12
8	Куб, 5	Площина, 4	Площина, 12	Згл. куб, 8	Згл. куб, 1	Згл. куб, 14
9	Куб, 12	Куб, 15	Куб, 6	Куб, 5	Куб, 2	Згл. куб, 14
10	Куб, 6	Згл. куб, 3	Куб, 4	Площина, 7	Куб, 14	Куб, 1
11	Площина, 11	Куб, 1	Площина, 7	Згл. куб, 15	Куб, 3	Куб, 14
12	Куб, 11	Згл. куб, 5	Площина, 12	Куб, 10	Згл. куб, 15	Куб, 1
13	Згл. куб, 2	Куб, 12	Куб, 2	Згл. куб, 15	Куб, 14	Куб, 7
14	Куб, 1	Куб, 6	Згл. куб, 8	Згл. куб, 3	Куб, 2	Згл. куб, 5
15	Згл. куб, 10	Площина, 12	Куб, 7	Куб, 4	Згл. куб, 2	Куб, 3
16	Площина, 4	Площина, 11	Куб, 10	Площина, 7	Куб, 5	Згл. куб, 8
17	Куб, 10	Куб, 3	Площина, 13	Куб, 6	Куб, 2	Площина, 12
18	Куб, 2	Куб, 1	Згл. куб, 8	Куб, 4	Площина, 7	Куб, 11
19	Площина, 11	Згл. куб, 2	Площина, 7	Куб, 5	Згл. куб, 15	Згл. куб, 3
20	Куб, 5	Куб, 11	Площина, 4	Згл. куб, 8	Куб, 10	Згл. куб, 15

Необхідно додати фігури відповідно до варіанту з таблиці А.2, параметри фігур взяти з таблиці А.1 за номером.

Назва класів фігур:

- площина – s_plane;
- куб – s_cube;
- згладжений куб – c_cube_smooth.

Додаток Б
Параметри джерел світла

№ варіанта	Колір фонового джерела (AAAA)	Колір спрямованого джерела (LLLL)	Направлення на джерело (DDD)
1	темно-червоний	білий	10.f, 20.f, 10.f
2	чорний	жовтий	-30.f, 30.f, 30.f
3	темно-зелений	червоний	10.f, 30.f, -10.f
4	темно-синій	білий	-30.f, 10.f, 0.f
5	темно-сірий	помаранчевий	-10.f, 10.f, 0.f
6	темно-синій	білий	-15.f, 3.f, 38.f
7	темно-зелений	світло сірий	10.f, 20.f, 10.f
8	темно-сірий	білий	-10.f, 50.f, -5.f
9	темно-сірий	ціан	-30.f, 10.f, 0.f
10	темно-синій	зелений	-15.f, 3.f, 38.f
11	темно червоний	ціан	-30.f, 30.f, 30.f
12	темно-сірий	помаранчевий	-30.f, 10.f, 0.f
13	чорний	ціан	-20.f, 30.f, 30.f
14	темно-синій	білий	-10.f, 50.f, -5.f
15	темно-зелений	зелений	-15.f, 3.f, 38.f
16	темно-сірий	світло сірий	-10.f, 10.f, 0.f
17	темно синій	білий	10.f, 30.f, -10.f
18	темно-червоний	зелений	-30.f, 10.f, 0.f
19	темно-синій	помаранчевий	10.f, 30.f, -10.f
20	темно-зелений	зелений	-10.f, 10.f, 0.f

Увага! Перед заданням вектора DDD його необхідно нормалізувати: розрахувати його довжину і кожну компоненту вектора розділити на неї.

Додаток В
Параметри відблисків

№ вар	Інтенсивність відблиску для фігур з парними номерами (SSS)	Інтенсивність відблиску для фігур з непарними номерами (SSS)	Розмір відблиску для фігур з парними номерами (PPP)	Розмір відблиску для фігур з непарними номерами (PPP)
1	0.5f	1.0f	128.f	32.f
2	0.25f	0.75f	32.f	120.f
3	1.0f	0.4f	64.f	4.f
4	0.75f	0.5f	96.f	16.f
5	0.25f	1.0f	4.f	128.f
6	0.4f	0.75f	16.f	80.f
7	0.5f	0.25f	8.f	16.f
8	1.0f	0.8f	32.f	64.f
9	0.8f	1.0f	80.f	128.f
10	0.75f	0.5f	56.f	10.f
11	0.25f	0.8f	64.f	32.f
12	0.8f	0.75f	128.f	20.f
13	1.0f	0.4f	15.f	100.f
14	0.5f	0.25f	28.f	80.f
15	0.8f	1.0f	64.f	35.f
16	0.75f	0.5f	3.f	120.f
17	0.3f	0.75f	10.f	80.f
18	0.3f	1.0f	30.f	96.f
19	0.5f	0.25f	70.f	10.f
20	0.25f	0.75f	28.f	89.f

Увага! Номери фігур для визначення парності та непарності брати з Таблиці 2 (додаток Б) лабораторної роботи № 5.

Додаток Г
Швидкості обертання

№ варіанта	Обертання RX	Обертання RY
1	30.f	0.f
2	-100.f	10.f
3	50.f	50.f
4	-20.f	-5.f
5	120.f	30.f
6	-5.f	-100.f
7	-20.f	0.f
8	10.f	10.f
9	-5.f	-20.f
10	30.f	-5.f
11	-5.f	120.f
12	-100.f	-5.f
13	10.f	50.f
14	0.f	10.f
15	-5.f	-20.f
16	-20.f	120.f
17	-5.f	30.f
18	50.f	-100.f
19	-5.f	0.f
20	120.f	10.f

Додаток Д
Швидкість, розмах зсуву джерела по осі Y, радіус освітлення

№ варіанта	Швидкість зміщення SSS	Розмах зміщення DDD	Радіус освітлення RRR
1	0.1f	35.f	50.f
2	0.2f	25.f	35.f
3	0.2f	25.f	45.f
4	0.05f	50.f	25.f
5	0.05f	45.f	50.f
6	0.1f	35.f	40.f
7	0.25f	25.f	25.f
8	0.05f	45.f	35.f
9	0.2f	50.f	40.f
10	0.25f	25.f	45.f
11	0.1f	45.f	40.f
12	0.05f	25.f	50.f
13	0.25f	35.f	35.f
14	0.05f	50.f	25.f
15	0.2f	50.f	45.f
16	0.25f	25.f	40.f
17	0.1f	45.f	50.f
18	0.05f	50.f	40.f
19	0.2f	35.f	35.f
20	0.1f	25.f	45.f

Список літератури

1. Гилой В. Интерактивная машинная графика : структуры данных, алгоритмы, языки / В. Гилой. — М. : Мир, 1981. — 380 с.
2. Дуда Р. Распознавание образов и анализ сцен : пер. с англ. / Р. Дуда, П. Харт. — М. : Мир, 1976. — Гл. 7, 9.
3. Ньюмен П. Основы интерактивной машинной графики / П. Ньюмен, Р. Спрулл ; пер. с англ. В. М. Грина, О. Н. Родинко. — М. : Мир, 1976. — 573 с.
4. Павлидис Т. Алгоритмы машинной графики и обработки изображений / Т. Павлидис. — М. : Радио и связь, 1986. — 198 с.
5. Порев В.Н. Компьютерная графика : учеб. пособие / Виктор Порев. — СПб. : ВHV-Санкт-Петербург, 2002. — 428 с.
6. Прэтт У. Цифровая обработка изображений : в 2 кн. / У. Прэтт ; пер. с англ. под ред. Д. С. Лебедева. — М. : Мир, 1982. — Гл. 2, 3, 12, 13, 17, 18, 20.
7. Роджерс Д. Алгоритмические основы машинной графики : пер. с англ. / Д. Роджерс. — М. : Мир, 1989. — 503 с.
8. Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. со 2-го англ. изд. П. А. Монахова [и др.]. — М. : Мир, 2001. — 604 с.
9. Тихомиров Ю. Программирование трехмерной графики : OpenGL, создание реалистических образов / Ю. Тихомиров. СПб. : ВHV-Санкт-Петербург, 1998. — 245 с.
10. Фоли Дж. Основы интерактивной машинной графики : в 2 кн. / Дж. Фоли. — М. : Мир, 1987. — Кн. 1. — 367 с. ; Кн. 2. — 365 с.
11. Шикин Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. — М. : ДИАЛОГ-МИФИ, 1995. — 286 с.
12. Шикин Е. В. Компьютерная графика. Полигональные модели / Е. В. Шикин, А. В. Боресков. — М. : Диалог-МИФИ, 2000. — 461 с.
13. Эйнджел Э. Интерактивная компьютерная графика : ввод. курс на базе OpenGL / Эдвард Эйнджел ; пер. с англ. и ред. В. Т. Тертышного. — 2-е изд. — М. : Вильямс, 2001. — 590 с.

Навчальне видання

ЗУЄВ Андрій Олександрович
ЄВСЕЄНКО Олег Миколайович
КРИЛОВА Вікторія Анатоліївна

КОМП'ЮТЕРНА ГРАФІКА.
ЧАСТИНА 2

Методичні вказівки

до виконання лабораторних робіт
для студентів спеціальностей
151 «Автоматизація та комп'ютерно-інтегровані технології»
172 «Телекомунікація та радіотехніка»

Відповідальний за випуск Качанов П.О.
Роботу до друку рекомендував О.В. Дудник

Редактор Самініна О.С.

План 2020 р., Поз. 59

Підписано до друку . Формат 60'84 1/16. Папір друк. № 2.
Друк - ризографія. Гарнітура Times New Roman. Умов. друк. арк. 3,2.
Обл.-вид. арк. 2,7. Наклад 100 прим. Зам. № . Ціна договірна.
