Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

А. А. Быков, Е. А. Мельникова, К. Д. Яшин

КОГНИТИВНЫЕ ТЕХНОЛОГИИ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия к практическим занятиям для специальности 1-58 01 01 «Инженерно-психологическое обеспечение информационных технологий»

Минск БГУИР 2015

УДК 004.81(076.5)
ББК 32.813я73
Б95

Р е ц е н з е н т ы:

кафедра программного обеспечения вычислительной техники и автоматизированных систем Белорусского национального технического университета (протокол №3 от 14 ноября 2013 г.);

ведущий научный сотрудник государственного научного учреждения «Объединенный институт проблем информатики Национальной академии наук Беларуси», доктор психологических наук Г. В. Лосик

Быков, А. А.

Б95 Когнитивные технологии : пособие / А. А. Быков, Е. А. Мельникова, К. Д. Яшин. – Минск : БГУИР, 2015. – 87 с. : ил.

ISBN 978-985-543-122-1.

Пособие содержит краткие теоретические сведения, методические указания к выполнению практических работ. Практические задания предназначены для студентов и преподавателей, владеющих теоретическим материалом по темам предлагаемых практических работ.

УДК 004.81(076.5)
ББК 32.813я73

ISBN 978-985-543-122-1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пособие содержит задания по практическому освоению способов разработки и эксплуатации систем визуализации данных в сеансе работы с виртуальной реальностью, проведению технической диагностики компьютерных систем трехмерной визуализации данных, разработке модулей систем виртуальной реальности типа тренажеров и автоматизированных рабочих мест.

Пособие поможет студентам в практическом освоении способов разработки системотехнических устройств, обеспечивающих трехмерное представление данных, визуальное представление их человеку и управление этими системотехническими средствами. Результатом проработки задач, представленных в настоящем пособии, является приобретение студентами навыков в области проектирования, создания и испытания сложных систем трехмерной визуализации.

Тема 1. Применение библиотеки Irrlight 3D для создания систем образной информации

Цель: формирование практических умений настройки систем трехмерной визуализации образной информации.

Теория и примеры выполнения задания

Рассмотрим графический движок (англ. graphics engine; иногда рендерер или визуализатор) – подпрограммное обеспечение (англ. middleware), программный движок для визуализации (рендеринга) двухмерной или трехмерной компьютерной графики. Графический движок может существовать как отдельный продукт или в составе игрового движка, может использоваться для визуализации отдельных изображений или компьютерного видео. Графические движки, использующиеся в программах по работе с компьютерной графикой (таких, как 3ds Max, Maya,Cinema 4D, Zbrush, Blender), обычно называются рендерерами, отрисовщиками или визуализаторами. Название «графический движок» используется, как правило, в компьютерных играх.

Физический движок (англ. physics engine) – компьютерная программа, которая производит симуляцию физических законов реального мира в виртуальном мире с той или иной степенью аппроксимации. Чаще всего физические движки используются не как отдельные самостоятельные программные продукты, а как составные компоненты (подпрограммы) других программ.

Для выполнения заданий мы будем использовать графический движок Irrlicht и библиотеку физики Crono::Engine.

Порядок выполнения работы:

1. Для установки и настройки графического движка необходимо настроить низкоуровневую библиотеку DirectX. Эта библиотека распространяется в виде исполняемого установочного файла: DXSDK_Jun10.exe.

2. Следующим этапом должна стать установка и настройка библиотек самого графического движка. Для этого необходимо распаковать архив irrlicht-1.7.2.zip с библиотеками Irrlicht в директорию D:\Work\3D. Также необходимо распаковать Библиотеки звука irrKlang-1.3.0b.zip.

3. Для установки физического движка необходимо запустить установочный файл ChronoEngine_v1.3.0.exe. По умолчанию он устанавливается в директрию c:\Program Files\ChronoEngine\.

4. В качестве шаблона можно запустить примеры, демонстрирующие работу Irrlicht и Crono::Engine. Готовые примеры Irrlicht можно найти в директории D:\Work\3D \irrlicht-1.7.2\bin\Win32-VisualStudio, примеры Crono::Engine – в директории c:\Program Files\ChronoEngine\bin\Win32_VisualStudio.

5. Для дальнейшей работы с библиотеками необходимо настроить MS VisualStudio 2008. Откройте d:\work\3d\irrlicht-1.7.2\examples. Здесь находятся исходные тексты примеров на C++, настроенные и готовые к компиляции и исполнению. Откройте /01.HelloWorld/ HelloWorld_vc9.vcproj. Это файл проекта первого примера вы можете увидеть окно на рис. 1. Он необходим для оптимизации проектов установленной у вас версии (если у вас установлена более поздняя версия VisualStudio).

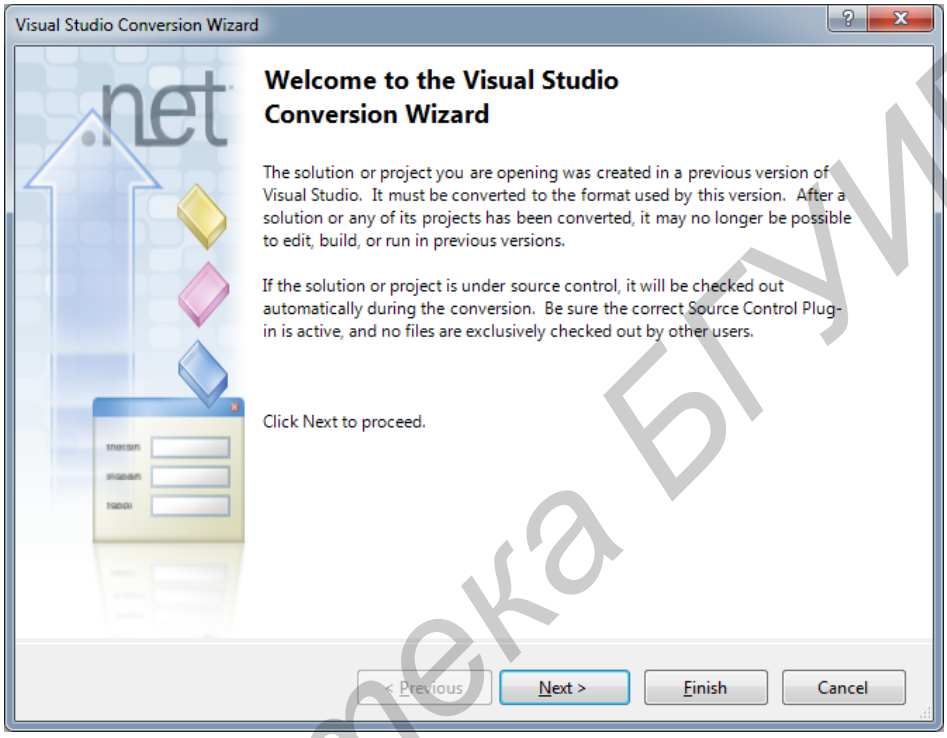

Рис.1. Приветственное окно Visual Studio

6. Текст программы находится в файле main.cpp (рис. 2). Окно Solution Explorer может быть свернуто в левой панели, наведите на свернутое окно мышь и кликните на него.

7. Для запуска необходимо кликнуть на зеленый треугольник в верхней части панели.

8. Для настройки проекта щелкните правой кнопкой мыши на элементе проекта в списке.

Получение готового exe файла происходит в два этапа:

- компиляция (Compile);
- сборка, линкование (Linker).

У этих объектов необходимо указать:

- для компилятора – заголовочные (*.h) файлы библиотеки Irrlicht;
- для сборщика – библиотеки (*.lib) Irrlicht – это путь к заголовочным файлам.

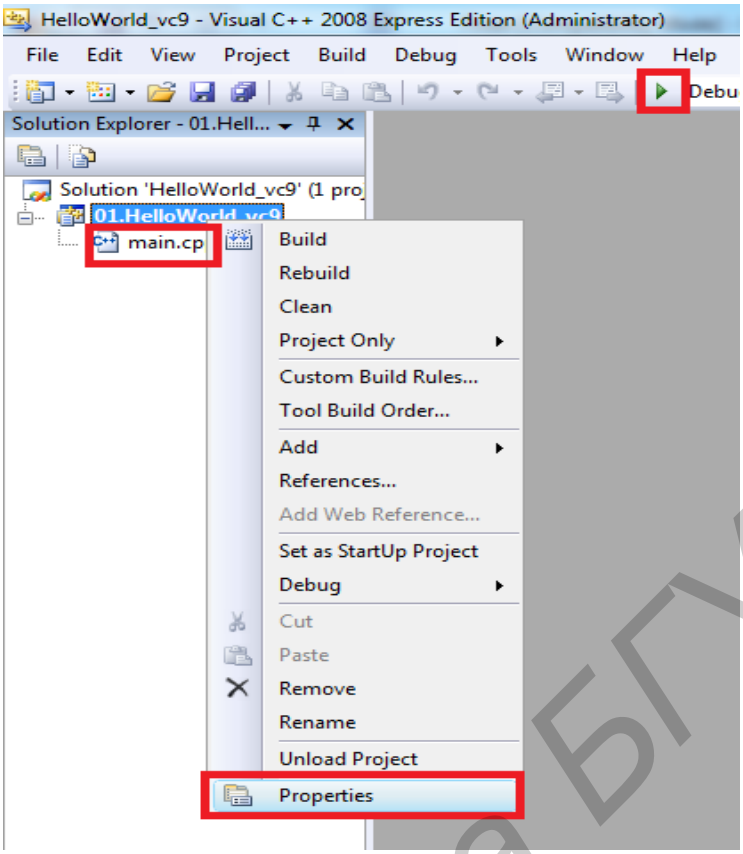

Рис. 2. Окно Solution Explorer

9. Таким же образом производится настройка библиотеки Crono::Engine (рис. 3). Исходный код можно найти в irrlicht_ChronoEngine_demo.zip.

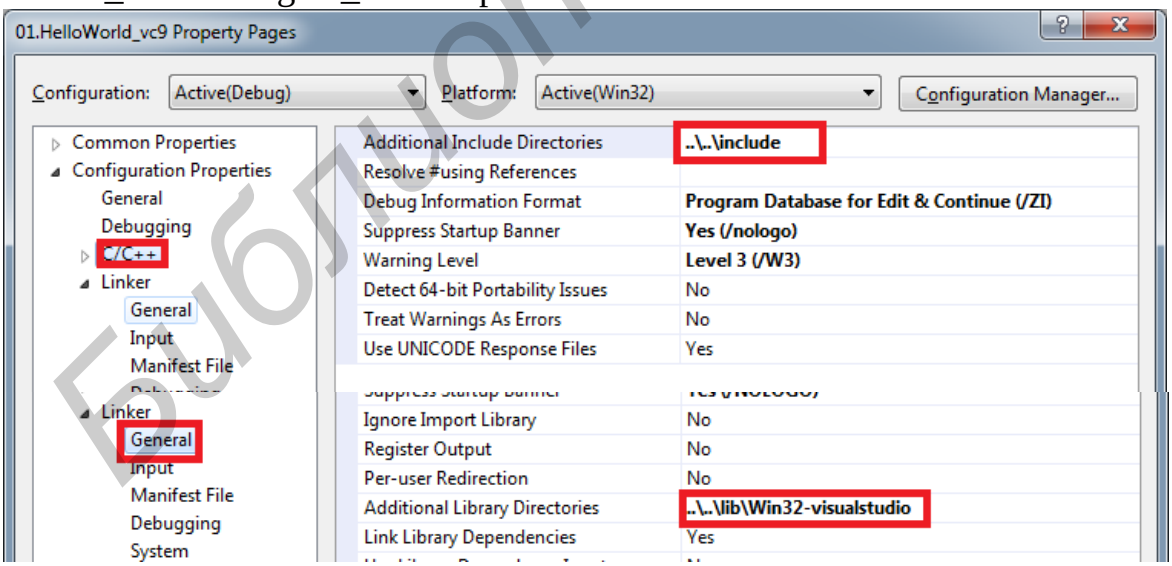

Рис. 3. Библиотека Crono::Engine

Задание: установить и настроить систему трехмерной визуализации образной информации.

Тема 2. Создание 3D сцен с использованием Irrlicht

Цель: формирование практических умений создания 3D сцен при помощи компьютерной библиотеки Irrlicht.

Теория и примеры выполнения заданий

Рассмотрим разработку приложения для визуализации 3D объектов. С этой целью необходимо настроить среду программирования VisualStudio (см. тему 1).

Порядок выполнения работы:

1. В гласном файле проекта main.cpp необходимо указать компилятору, что далее будем использовать объекты и функции библиотеки IrrLicht, это выполняется так:

```
#include < irrlicht.h >
#include < driverChoice.h >
```

Если при компиляции среда печатает ошибки, связанные с этими двумя строчками, необходимо правильно указать компилятору пути к заголовочным файлам библиотеки. Все функции и объекты IrrLight определены в пространстве имен (namespace) 'irr'. Таким образом, если надо обратиться к классу движка, то надо написать irr::, перед его именем.

```
using namespace irr;
```

2. Ниже перечислены пять подпространств движка. Подробное описание каждого из них можно узнать, прочитав документацию, поставляемую вместе с IrrLight. Как и в случае с irr::, можно указать, что они будут использоваться следующим образом:

```
using namespace core;
using namespace scene;
using namespace video;
using namespace io;
using namespace gui.
```

3. Чтобы использовать Irrlicht.DLL, надо добавить библиотеку Irrlicht.lib в проект. Это можно сделать двумя способами: указать в настройках линкера или использовать директиву pragma comment.

```
#ifdef _IRR_WINDOWS_
#pragma comment(lib, "Irrlicht.lib")
```

```
#pragma comment(linker, "/subsystem:windows
/ENTRY:mainCRTStartup")
#endi
```

4. Для того чтобы приложение было кросс-платформенным, будет использовать main вместо WinMain.

Самая главная функция движка – это createDevice().

5. При помощи этой функции создается корневой объект движка IrrlichtDevice. Функция createDevice() имеет 7 аргументов:

- deviceType: Тип устройства. Принимает значения: EDT_SOFTWARE, EDT_BURNINGSVIDEO, EDT_NULL, EDT_DIRECT3D8, EDT_DIRECT3D9 или EDT_OPENGL;
- windowSize: размер окна в оконном режиме или разрешение экрана в полноэкранном режиме;
- bits: количество бит для цвета на один пиксель; может быть 16 или 32; параметр часто игнорируется в оконном режиме;
- fullscreen: определяет в оконном или полноэкранном режиме, будет ли запущено приложение;
- stencilbuffer: включает трафаретный буфер (для рисования теней);
- vsync: включает вертикальную синхронизацию, полезна для полноэкранного режима;
- eventReceiver: объект – обработчик событий.

6. Перед началом работы необходимо получить обратную информацию – какой видеодрайвер необходимо использовать:

```
// спрашиваем у пользователя про дарайвер
// (DirectX, OpenGL и т.п.)
video::E_DRIVER_TYPE driverType = driverChoiceConsole();
IrrlichtDevice *device = createDevice(
     video::EDT_SOFTWARE, dimension2d< u32 >(640, 480),
     16, false, false, false, 0);
 if (!device) return 1;
```

7. Далее необходимо определить заголовок окна:

```
device->setWindowCaption(
            L"Hello World! - Irrlicht Engine Demo");
```

8. Затем создаем видеодрайвер, менеджер сцены и менеджер пользовательского интерфейса:

```
IVideoDriver* driver = device->getVideoDriver();
```

```
ISceneManager* smgr = device->getSceneManager();
IGUIEnvironment* guienv = device->getGUIEnvironment();
```

9. С помощью GUI среды можно создать текстовую метку в координатах (10,10) левый верхний угол и (260,22) правый нижний. Чтобы выводимый текст был на русском языке, файлы проекта должны быть в кодировке UTF-8.

```
guienv->addStaticText(L"Hello World!
      This is the Irrlicht Software renderer!",
      rect< s32 >(10,10,260,22), true);
```

10. Для демонстрации возможностей IrrLight загрузим модель из игры Quake2 и отобразим ее. Чтобы избежать ошибок, необходимо учитывать начальный каталог для приложения. Если запускаете пример при помощи VisualStudio, начальным каталогом является каталог, где сохранен проект. Если запускаете exe файл, начальным каталогом является каталог, где находится exe файл. Это важно потому, что по умолчанию exe файл записывается в каталог debug.

Метод getMesh универсальный и хранит информациию о всех типах, которые можно загрузить в движок, список поддерживаемых форматов можно найти на официальном сайте IrrLight.

```
IAnimatedMesh* mesh = smgr->
      getMesh("../../media/sydney.md2");
if (!mesh)
{
      device->drop();
      return 1;
}
IAnimatedMeshSceneNode* node = smgr->
      addAnimatedMeshSceneNode( mesh );
```

11. Чтобы модель выглядела привлекательно, надо включить в нее ntrcnehe. Освещение в данном случае необходимо отключить потому, что она будет рисоваться абсолютно черной. Это происходит потому, что на сцене нет источников света. Также определим диапазон фреймов, анимацию, которая будет воспроизводиться. Для реализации данных операций используется набор STAND из списка стандартных анимаций для моделей Quake2.

```
if (node) {
      node->setMaterialFlag(EMF_LIGHTING, false);
      node->setMD2Animation(scene::EMAT_STAND);
```

```
        node->setMaterialTexture( 0, driver->
                getTexture("../../media/sydney.bmp") );
    }
```

12. Чтобы получить результат, необходимо добавить на сцену камеру и направить ее на модель. Камеру разместим в позиции (0, 30, -40) и направим в точку (0,5,0), где приблизительно располагается загруженная модель.

```
smgr->addCameraSceneNode(0, vector3df(0,30,-40),
                                vector3df(0,5,0));
```

13. Далее запускаем главный цикл программы, в котором происходит обрисовка:

```
while(device->run()) {
```

Обрисовка происходит между вызовами beginScene() и endScene(). Функция beginScene() закрашивает выбранным цветом основной буфер экрана и дополнительный (если нужно). Затем даются команды менеджеру сцены и менеджеру пользовательского интерфейса нарисовать свой образ. Наконец функция endScene() выводит все на экран.

```
driver->beginScene(true, true, SColor(255,100,101,140));
smgr->drawAll();
guienv->drawAll();
driver->endScene();
```

14. По завершении главного цикла программы удаляется главный объект движка, созданный с помощью createDevice(). В движке Irrlicht все объекты, созданные функциями, начинающимися с приставки 'create', можно удалить методом **drop**.

```
device->drop();
return 0;
```

Задание: создать программу визуализации 3D объекта.

Тема 3. Управление объектами и взором компьютерных тренажеров

Цель: формирование практических умений настройки органов управления компьютерных тренажеров.

Теория и примеры выполнения задания

Рассмотрим процесс перемещения (node) сцены с помощью клавиатуры и анимированных узлов.

Порядок выполнения работы:

1. Подключаем заголовочный файл IrrLicht, объявляем пространство имен 'irr'.

```
#include < irrlicht.h >
#include "driverChoice.h"
using namespace irr;
```

Для обработки таких событий, как события мыши, клавиатуры, пользовательского интерфейса, необходимо реализовать класс, унаследованный от irr::IEventReceiver и переопределить его метод irr::IEventReceiver::OnEvent(), потом создать объект на основе своего класса, «сообщить» о нем движку, который в свою очередь каждый раз, когда в нем возникает событие, будет вызывать наш обработчик OnEvent.

```
class MyEventReceiver : public IEventReceiver {
public:
// наш собственный обработчик событий
 virtual bool OnEvent(const SEvent& event) {
       // просто запоминаем состояние любой
       // клавиши - нажата/отжата
       if (event.EventType == irr::EET_KEY_INPUT_EVENT)
             KeyIsDown[event.KeyInput.Key] =
                         event.KeyInput.PressedDown;
       return false;
 }
  // метод, возвращающий состояние для запрошенной клавиши
 virtual bool IsKeyDown(EKEY_CODE keyCode) const {
       return KeyIsDown[keyCode];
 }
 //конструктор, в цикле сбрасываем статус для всех клавиш
 MyEventReceiver() {
       for (u32 i=0; i<KEY_KEY_CODES_COUNT; ++i)
             KeyIsDown[i] = false;
 }
private:
```

```
    // массив для хранения статусов клавиш
    bool KeyIsDown[KEY_KEY_CODES_COUNT];
};
private:
    // массив для хранения статусов клавиш
    bool KeyIsDown[KEY_KEY_CODES_COUNT];
};
```

2. Обработчик событий во время работы главного цикла будет запоминать, какая клавиша нажата, и выполнять определенные для нее действия. Далее создадим корневой объект (движок) irr::IrrlichtDevice и узел, который будем двигать по сцене. Также создадим еще несколько узлов, чтобы показать различные способы для перемещения объектов на сцене.

```
int main()
{
// спрашиваем у пользователя при запуске про драйвер
// (OpenGL, DirectX и т.п.)
 video::E_DRIVER_TYPE driverType=driverChoiceConsole();
 if (driverType==video::EDT_COUNT)
      // выходим из приложения, если драйвер не выбран
      return 1;
// создаем наш обработчик событий
 MyEventReceiver receiver;
// при создании движка сообщаем про наш обработчик,
// см. последний параметр.
 IrrlichtDevice* device = createDevice(driverType,
     core::dimension2d< u32 >(640, 480), 16, false,
     false, false, &receiver);
 if (device == 0)
      return 1; // ошибка создания движка, выходим.
 // вспомогательные указатели
 video::IVideoDriver* driver = device->getVideoDriver();
 scene::ISceneManager* smgr = device->getSceneManager();
```

3. Создаем узел, который будет реагировать на нажатие клавиш WASD. Это будет сфера, созданная с помощью встроенной функции. Разместим ее в

координатах (0,0,30) и наведем на нее текстуру. Так как на сцене нет динамических источников света, то для всех материалов, привязываемых к узлам сцены, отключаем обработку освещения, иначе они все будут абсолютно черными.

```
scene::ISceneNode * node = smgr->addSphereSceneNode();
// создаем на сцене сферу
if (node)
{
      // позиционируем сферу
      node->setPosition(core::vector3df(0,0,30));
      // текстурируем сферу
      node->setMaterialTexture(0, driver->
                  getTexture("../../media/wall.bmp"));
      //отключаем обработку освещения для материала
      node->setMaterialFlag(video::EMF_LIGHTING, false);
 }
```

4. Теперь создаем другой узел, который будет двигать аниматор. Аниматоры могут влиять на узлы сцены, к которым привязаны, например, 3D модели, билборды, источники света, камеры. Аниматоры могут влиять на практически любые характеристики узла, т. е. не только двигать, но и масштабировать объекты, анимировать текстуры и т.п. Создадим куб и присоединим к нему аниматор, который заставит его летать по кругу.

```
// создаем на сцене куб
scene::ISceneNode* n = smgr->addCubeSceneNode();
if (n) {
      // текстурируем куб
      n->setMaterialTexture(0, driver->
                  getTexture("../../media/t351sml.jpg"));
      // отключаем обработку освещения
      n->setMaterialFlag(video::EMF_LIGHTING, false);
      // позиционировать не будем, т.к. его спозиционирует
      // аниматор, созданный в следующей строке
      scene::ISceneNodeAnimator* anim = smgr->
            createFlyCircleAnimator(
                        core::vector3df(0,0,30), 20.0f);
```

```
        if (anim) {
                n->addAnimator(anim);
                anim->drop();
        }
    }
```

5. Есть еще один узел на сцене, в которой мы анимируем 3D модель ниндзи, используя аниматор движения по прямой между двух точек.

```
    scene::IAnimatedMeshSceneNode* anms = smgr->
        addAnimatedMeshSceneNode(smgr->
                    getMesh("../../media/ninja.b3d"));
    if (anms) {
        scene::ISceneNodeAnimator* anim = smgr->
                createFlyStraightAnimator(
                core::vector3df(100,0,60),
                core::vector3df(-100,0,60), 3500, true);
        if (anim) {
                anms->addAnimator(anim);
                anim->drop();
        }
```

Чтобы ниндзя двигался, аниматор доставляет его из точки А в точку Б. Мы дадим указание на проигрывание анимационного набора фреймов модели, отвечающего за бег. Чтобы правильно сориентировать модель, развернем ее на 90° и чуть-чуть отмасштабируем. Если бы мы использовали модель .md2 вместо .b3d, то и для задания диапазона фремов вместо setFrameLoop() и setAnimationSpeed() можно было бы вызвать "anms->setMD2Animation(scene::EMAT_RUN)", т. к. для моделей из Quake2 в движке есть предопределенные константы.

```
    anms->setMaterialFlag(video::EMF_LIGHTING, false);
    // фремы модели с 0 по 13 изображают бег
    anms->setFrameLoop(0, 13);
    // скорость проигрывания анимации
    anms->setAnimationSpeed(15);
    // маштабируем ниндзю в 2 раза по XYZ осям
    anms->setScale(core::vector3df(2.f,2.f,2.f));
    // поворачиваем ниндзю на 90° вокруг Y
```

```
anms->setRotation(core::vector3df(0,-90,0));
```

6. Чтобы управлять камерой при помощи мыши, добавим на нее FPS камеру и удалим указатель мыши.

```
0,  smgr->addCameraSceneNodeFPS();
 device->getCursorControl()->setVisible(false);
```

7. После того как создали все необходимые для анимации объекты, рассмотрим обрисовку сцены:

```
int lastFPS = -1;
// чтобы сделать жизнь сцены независимой от частоты кадров
// мы должны знать время, прошедшее с момента отрисовки
// последнего фрейма
u32 then = device->getTimer()->getTime();
// расстояние, покрываемое объектами в секунду.
const f32 MOVEMENT_SPEED = 5.f;
while(device->run())
{
      // ищем разницу во времени
      const u32 now = device->getTimer()->getTime();
      // в секундах
      const f32 frameDeltaTime = (f32)(now - then)
                                          / 1000.f;
      then = now;
```

8. Проверяем, нажаты ли клавиши W, S, A, D. Если да, то двигаем сферу.

```
// получаем текущую позицию сферы
core::vector3df nodePosition = node->getPosition();
// двигаем текущую позицию вперед или назад
if(receiver.IsKeyDown(irr::KEY_KEY_W))
      nodePosition.Y
                  += MOVEMENT_SPEED * frameDeltaTime;
else if(receiver.IsKeyDown(irr::KEY_KEY_S))
      nodePosition.Y
                  -= MOVEMENT_SPEED * frameDeltaTime;
// двигаем ее текущую позицию вправо или влево
```

```
if(receiver.IsKeyDown(irr::KEY_KEY_A))
        nodePosition.X
                    -= MOVEMENT_SPEED * frameDeltaTime;
else if(receiver.IsKeyDown(irr::KEY_KEY_D))
        nodePosition.X
                    += MOVEMENT_SPEED * frameDeltaTime;
// применяем измененную позицию обратно к сфере
node->setPosition(nodePosition);
driver->beginScene(true, true,
                           video::SColor(255,113,113,133));
smgr->drawAll(); // рисуем объекты на сцене
// рисуем пользовательский интерфейс
device->getGUIEnvironment()->drawAll();
driver->endScene();
int fps = driver->getFPS();
if (lastFPS != fps) {
        // строка заголовка окна
        core::stringw tmp(L"Movement Example - Irrlicht
        Engine [");
        // к которому приплюсовываем имя драйвера
        tmp += driver->getName();
        // и количество рисуемых кадров в секунду
        tmp += L"] fps: ";
        tmp += fps;
        // устанавливаем заголовок
        device->setWindowCaption(tmp.c_str());
        lastFPS = fps;
}
```

9. По завершении главного цикла удаляем корневой объект (движок).

```
device->drop();
return 0;
```

Задание: создать программу для управления своим собственным объектом.

Тема 4. Создание эффекта стереопсиса

Цель: формирование практических умений создания эффекта стереопсиса.

Теория и примеры выполнения задания

Для создания эффекта стереопсиса необходимо выводить различное изображение для каждого из глаз. Стереомонитор использует возможности видеокарты компьютера и позволяет использовать 2 рабочих стола Windows, расположенных горизонтально рядом.

Порядок выполнения работы

1. Для работы будем использовать свойство, по которому рабочие столы используют общее пространство. Так, окно можно располагать частично в пространстве первого рабочего стола, частично – второго. Таким образом, для того чтобы использовать стереомонитор для создания эффекта стереопсиса, необходимо задать размер окна в 2 размера рабочего стола и обрисовывать 2 изображения в IrrLicht Engine, разделенных вертикально.

2. После того как обрисовали заголовки (#include), определим глобальные константы, описывающие разрешение основного окна для использования при инициализации устройства и установки областей просмотра. Установим глобальные переменные, указывающие, активен или нет режим разбиения экрана.

```
//Разрешение экрана
const int ResX=800;
const int ResY=600;
const bool fullScreen=false;
//по умолчанию используем режим SplitScreen
bool SplitScreen=true;
```

3. Теперь нам необходимы 4 указателя на наши камеры, которые будут созданы позже:

```
//камеры
ICameraSceneNode *camera[3]={0,0,0};
```

В нашем event-receiver (получальщике событий) переключаем SplitScreen переменные каждый раз, когда пользователь нажимает клавишу «S». Все остальные события посылаются FPS камере.

```
class MyEventReceiver : public IEventReceiver {
public:
virtual bool OnEvent(const SEvent& event) {
```

```
        //Клавиша S включает/выключает режим SplitScreen
        if (event.EventType == irr::EET_KEY_INPUT_EVENT &&
                    event.KeyInput.Key == KEY_KEY_S &&
                    event.KeyInput.PressedDown) {
              SplitScreen = !SplitScreen;
              return true;
        }
        //Послать все остальные события оставшимся 4 камерам
        if (camera[2])
              return camera[2]->OnEvent(event);
              return false;
        }
  };
```

4. Теперь рассмотрим основную функцию: инициируем устройство, получаем менеджер сцены и видеодрайвер, загружаем анимированный меш из .md2 и карту из .pk3.

```
  int main(int argc, char** argv)
  {
        // спросить пользователя о драйвере
        // (OpenGL, DirectX и т.п.)
        video::E_DRIVER_TYPE driverType =
                                driverChoiceConsole();
        if (driverType==video::EDT_COUNT)
              driverType = video::EDT_OPENGL;

        // Создать экземпляр EventReceiver
        // (обработчика событий)
        MyEventReceiver receiver;
        // Инициализация движка
        IrrlichtDevice *device = createDevice(driverType,
              dimension2du(ResX,ResY), 32, fullScreen,
                                false, false, &receiver);
        if (!device)
              return 1;
```

```
ISceneManager *smgr = device->getSceneManager();
IVideoDriver *driver = device->getVideoDriver();
// Загрузить модель
IAnimatedMesh *model = smgr->
            getMesh("../../media/sydney.md2");
if (!model)
      return 1;
IAnimatedMeshSceneNode *model_node = smgr->
            addAnimatedMeshSceneNode(model);
// Загрузить текстуру
if (model_node) {
      ITexture *texture = driver->
            getTexture("../../media/sydney.bmp");
      model_node->setMaterialTexture(0,texture);
      model_node->setMD2Animation(scene::EMAT_RUN);
      // отключаем освещение
      model_node->
            setMaterialFlag(EMF_LIGHTING,false);
}
// Загрузить карту
device->getFileSystem() -> addZipFileArchive(
                  "../../media/map-20kdm2.pk3");
IAnimatedMesh *map = smgr->getMesh("20kdm2.bsp");
if (map) {
      ISceneNode *map_node = smgr->
            addOctreeSceneNode(map->getMesh(0));
      // установить позицию
      map_node->setPosition(
                  vector3df(-850,-220,-850))
```

5. Создадим 3 FPS камеры: одна будет управлять пользователем, две будут смотреть параллельно в том же направлении, что и управляемая пользователем.

```
// Создать 3 фиксированные
// и 1 управляемую пользователем камеру
// Вид спереди
```

```
camera[2] = smgr->addCameraSceneNode(0, vector3df(50,0,0),
                                     vector3df(0,0,0));
if (!camera[2])
      return 1;
//Правая
camera[0] = smgr->addCameraSceneNodeFPS();
if (camera[0])
      camera[0]->setPosition(vector3df(50,0,-0.01f));
//Левая
camera[1] = smgr->addCameraSceneNodeFPS();
if (camera[1])
      camera[1]->setPosition(vector3df(50,0,0.01f));
// Создать переменную для подсчета fps(фреймов в секунду)
// и скрыть мышь:
device->getCursorControl()->setVisible(false);
// тут будет подсчитывать fps
int lastFPS = -1;
```

6. Создав 3 камеры, мы не добьемся того, чтобы экран был разбит/разделен. Чтобы это сделать, потребуется несколько шагов:

– установить область просмотра на весь экран;

– создать новую сцену (очистить экран);

– следующие 3 шага повторяются для каждой области просмотра на разделенном экране:

– установить область просмотра в необходимую вам область;

– активировать камеру, «связанную» с областью просмотра;

– визуализировать (отрендерить) все объекты.

Если у вас есть GUI, необходимо:

– установить область просмотра на весь экран;

– отобразить (отрисовать) GUI.

End scene – команда конца отрисовки сцены в игровом цикле.

Далее необходимо направить стереоскопические камеры для левого и правого глаза в том же направлении, что и камеры, управляемые пользователем (FPS).

```
while(device->run()) {
      // Установить область просмотра на весь экран
      // и запустить сцену
      driver->setViewPort(rect(0,0,ResX,ResY));
```

```
        driver->beginScene(true,true,
                           SColor(255,100,100,100));
        // Если режим SplitScreen включен
    if (SplitScreen) {
        // Это камера, управляемая пользователем
        // Получаем ее текущую позицию и точку взора
        vector3df oldPosition = camera[2]->getPosition();
        vector3df oldTarget = camera[2]->getTarget();
        matrix4 startMatrix = camera[2]->
                           getAbsoluteTransformation();
        vector3df focusPoint = (camera[2]->getTarget() -
        camera[2]->getAbsolutePosition()).setLength(1) +
        camera[2]->getAbsolutePosition() ;
        // Активировать камеру, управляемую пользователем
        smgr->setActiveCamera(camera[2]);
        // Установить область отрисовки неактивной камеры
        // размером в одну точку
        driver->setViewPort(rect<s32>(0,0,1,1));
        // Отрисовать сцену
        smgr->drawAll();
        //Левый глаз
        vector3df leftEye;
        matrix4   leftMove;
        leftMove.setTranslation(
                           vector3df(-0.01f,0.0f,0.0f));
        leftEye=(startMatrix*leftMove).getTranslation();
        // Устанавливаем позицию взора для левого глаза
        camera[0]->setPosition( leftEye );
        camera[0]->setTarget( focusPoint );
        // Активировать камеру левого глаза
        smgr->setActiveCamera(camera[0]);
        // Установить область просмотра в левой половине
        driver->setViewPort(rect<s32>(0,0,ResX/2,ResY));
        // Отрисовать сцену
        smgr->drawAll();
```

```
        //Такие же операции необходимо выполнить для камеры,
        //отображающей изображение для правого глаза
        * * *
}else{
        // Активировать камеру, управляемую пользователем
        smgr->setActiveCamera(camera[2]);
        driver->setViewPort(rect<s32>(0,0,ResX,ResY));
        // Отрисовать сцену
        smgr->drawAll();
}
driver->endScene();
```

7. Видно, что изображение отрисовывается для каждой области просмотра по отдельности. Затем создаем прямоугольник, определяя 4 координаты:

X – координата левого верхнего угла;
Y – координата левого верхнего угла;
X – координата правого нижнего угла;
Y – координата правого нижнего угла.

То есть если вы желаете разбить экран на 2 области просмотра, вы могли бы задать следующие координаты:

1-я область просмотра: 0,0,ResX/2,ResY;
2-я область просмотра: ResX/2,0,ResX,ResY.

8. Выключим движок

```
// получить и отобразить fps
if (driver->getFPS() != lastFPS) {
        lastFPS = driver->getFPS();
        core::stringw tmp = L"Irrlicht SplitScreen-Example”
        tmp +=”(FPS:” +lastFPS+ ")";
        device->setWindowCaption(tmp.c_str());
}
// Удалить устройство
 device->drop();
 return 0;
}
```

Примечание. Нажимая клавишу «S», можно включать и выключать режим splitscreen.

Задание: создать программу для стереоскопической визуализации своего собственного объекта.

Тема 5. Органы взаимодействия с тренажером

Цель: формирование практических умений при работе с джойстиком для управления 3D сценами.

Теория и примеры выполнения задания

На этом практическом занятии мы научимся обрабатывать события мыши и джойстика, если у вас подключен джойстик и имеется его поддержка. В настоящее время такая поддержка встроена в Windows, Linux и SDL устройства.

Мы будем сохранять последние состояния мыши и первого джойстика, при помощи слушателя обновлять данные, как только получаем события.

```
class MyEventReceiver : public IEventReceiver
{
public:
// Создаем структуру для записи состояний мыши
struct SMouseState
{
    core::position2di Position;
    bool LeftButtonDown;
    SMouseState() : LeftButtonDown(false) { }
} MouseState;

// Это единственный метод, который мы должны переопределить
virtual bool OnEvent(const SEvent& event)
{
// Запомнить состояния мыши
if (event.EventType == irr::EET_MOUSE_INPUT_EVENT)
{
       switch(event.MouseInput.Event)
{
       case EMIE_LMOUSE_PRESSED_DOWN:
```

```
            MouseState.LeftButtonDown = true;
        break;

            case EMIE_LMOUSE_LEFT_UP:
            MouseState.LeftButtonDown = false;
        break;

            case EMIE_MOUSE_MOVED:
            MouseState.Position.X = event.MouseInput.X;
            MouseState.Position.Y =
        break;

            default:
        // Мы не будем обрабатывать колесико мыши
        break;
        }
        }

        // Состояние каждого подключенного джойстика направляется
сюда
        // при каждом шаге главного цикла движка Irrlicht Engine
        // сохраняем состояние первого джойстика, игнорируем другие
джойстики.
        // Это на текущий момент поддерживается только в Windows и
Linux.
        if (event.EventType == irr::EET_JOYSTICK_INPUT_EVENT
        && event.JoystickEvent.Joystick == 0) {
        JoystickState = event.JoystickEvent;
        }
        return false;
        }
        const SEvent::SJoystickEvent & GetJoystickState(void) const
        {
            return JoystickState;
        }
```

```
const SMouseState & GetMouseState(void) const
{
	return MouseState;
}
MyEventReceiver()
{
}

private:
SEvent::SJoystickEvent JoystickState;
};
```

Обработчик событий для сохранения нажатых кнопок готов, ответная реакция будет добавлена внутри цикла приложения, перед отрисовкой сцены. Так что давайте просто создадим irr::IrrlichtDevice и узлы сцены, которые желаем двигать. Мы также создадим несколько дополнительных объектов, чтобы показать, что есть несколько дополнительных возможностей для перемещения и анимации узлов сцены.

```
int main()
{
// спросить пользователя про драйвер
video::E_DRIVER_TYPE driverType=driverChoiceConsole();
if (driverType==video::EDT_COUNT) return 1;

// создаем обработчик событий
MyEventReceiver receiver;

IrrlichtDevice* device = createDevice(driverType,
	core::dimension2d< u32 >(640, 480), 16, false, false,
false, &receiver);

if (device == 0)
	return 1; // не удалось создать выбранный драйвер.

core::array< SJoystickInfo > joystickInfo;
// если поддержка джойстика имеется
if(device->activateJoysticks(joystickInfo))
```

```
    {
          std::cout << "Joystick support is enabled and " <<
joystickInfo.size()
    << " joystick(s) are present." << std::endl;

    for(u32 joystick = 0; joystick < joystickInfo.size();
++joystick)
    {
    std::cout << "Joystick " << joystick << ":" << std::endl;
    std::cout << "tName: '" <<
joystickInfo[joystick].Name.c_str() << "'" << std::endl;
    std::cout << "tAxes: " << joystickInfo[joystick].Axes <<
std::endl;
    std::cout << "tButtons: " << joystickInfo[joystick].Buttons
<< std::endl;
    std::cout << "tHat is: ";

switch(joystickInfo[joystick].PovHat)
    {
    case SJoystickInfo::POV_HAT_PRESENT:
    std::cout << "present" << std::endl;
    break;

    case SJoystickInfo::POV_HAT_ABSENT:
    std::cout << "absent" << std::endl;
    break;

    case SJoystickInfo::POV_HAT_UNKNOWN:
    default:
    std::cout << "unknown" << std::endl;
    break;
    }
    }
    }
```

```
    else // джойстик не поддерживается
    {
    std::cout << "Joystick support is not enabled." <<
std::endl;
    }

    core::stringw tmp = L"Irrlicht Joystick Example (";
    tmp += joystickInfo.size();
    tmp += " joysticks)";
    device->setWindowCaption(tmp.c_str());

    video::IVideoDriver* driver = device->getVideoDriver();
    scene::ISceneManager* smgr = device->getSceneManager();
```

Мы создадим стрелку (3D модель), ось которой будет представлять импровизированную ручку джойстика и реагировать на отклонения реального джойстика, вдобавок эта «ручка» будет перемещаться вслед за указателем мыши.

```
    scene::ISceneNode * node = smgr->addMeshSceneNode(
    smgr->addArrowMesh( "Arrow",
          video::SColor(255, 255, 0, 0),
          video::SColor(255, 0, 255, 0),
                16,16,
                2.f, 1.3f,
                0.1f, 0.6f
                )
          );
    node->setMaterialFlag(video::EMF_LIGHTING, false);

    scene::ICameraSceneNode * camera = smgr-
>addCameraSceneNode();
    camera->setPosition(core::vector3df(0, 0, -10));

    // Как и в примере 04, мы сделаем движение независимым от
частоты кадров.
```

```
	u32 then = device->getTimer()->getTime();
	const f32 MOVEMENT_SPEED = 5.f;

	while(device->run())
	{
	// Рассчитаем временную разницу(DeltaTime).
	const u32 now = device->getTimer()->getTime();
	const f32 frameDeltaTime = (f32)(now - then) / 1000.f; // время в секундах
	then = now;

	bool movedWithJoystick = false;
	core::vector3df nodePosition = node->getPosition();

	if(joystickInfo.size() > 0)
	{
		f32 moveHorizontal = 0.f;
	// Диапазон от -1.f (полное влево) до +1.f (полное вправо)
		f32 moveVertical = 0.f; // от -1.f (полное вниз) до +1.f (полное вверх).

		const SEvent::SJoystickEvent & joystickData = receiver.GetJoystickState();

	// Мы получаем полный аналог диапазона осей(т.е. текущий наклон джойстика),
	// и таким образом можем определить мертвую зону. Это эмпирическое значение, так как
	// некоторые джойстики могут дрожать или ползать вокруг центральной точки больше других.
	// Мы используем диапазон 5% как мертвую зону, но в целом было бы неплохо, если бы вы
	// представили пользователю опциональную возможность изменять этот параметр.
```

```
	const f32 DEAD_ZONE = 0.05f;

	moveHorizontal =

(f32)joystickData.Axis[SEvent::SJoystickEvent::AXIS_X] / 32767.f;
	if(fabs(moveHorizontal) < DEAD_ZONE)
	 moveHorizontal = 0.f;

	moveVertical =

 (f32)joystickData.Axis[SEvent::SJoystickEvent::AXIS_Y] / -
 32767.f;
	if(fabs(moveVertical) < DEAD_ZONE)
		moveVertical = 0.f;

	// Информация о POV(Point of View) шляпки(т.е. шляпки-
 манипулятора дж.)
	// на данный момент поддерживается только в Windows, но
 значение
	// устанавливается в 65535, если это не поддерживается так,
 что мы можем
	// проверить этот диапазон.
	const u16 povDegrees = joystickData.POV / 100;
	if(povDegrees < 360)
	{
	if(povDegrees > 0 && povDegrees < 180)
		moveHorizontal = 1.f;

	if(povDegrees > 90 && povDegrees < 270)
		moveVertical = -1.f;

else if(povDegrees > 270 || povDegrees < 90)
		moveVertical = +1.f;
	}
```

```
    if(!core::equals(moveHorizontal, 0.f) ||
!core::equals(moveVertical, 0.f))
    {
          nodePosition.X += MOVEMENT_SPEED * frameDeltaTime *
moveHorizontal;
          nodePosition.Y += MOVEMENT_SPEED * frameDeltaTime *
moveVertical;
                movedWithJoystick = true;
          }
    }

    // Если нод-стрелка не перемещается джойстиком, она должна
следовать за курсором мыши.
    if(!movedWithJoystick)
    {
    // Создадим луч на курсор мыши.
    core::line3df ray = smgr->getSceneCollisionManager()-
>getRayFromScreenCoordinates(
receiver.GetMouseState().Position, camera);

    // Построим пересечение луча с плоскостью, обращенной к
камере,
    // расположенной вокруг нода(т. е. нод расположен на этой
плоскости,
    // так как плоскость строится в позиции нода).
    core::plane3df plane(nodePosition, core::vector3df(0, 0, -
1));
    core::vector3df mousePosition;
    if(plane.getIntersectionWithLine(ray.start,
ray.getVector(), mousePosition))
    {
    // Теперь мы имеем позицию мыши в 3D пространстве, двигаем
нод к этой позиции.
```

```
        core::vector3df toMousePosition(mousePosition -
 nodePosition);
        const f32 availableMovement = MOVEMENT_SPEED *
 frameDeltaTime;

        if(toMousePosition.getLength() <= availableMovement)
        nodePosition = mousePosition;
        // Прыгнуть к финальной позиции
 else // иначе двигаться к ней
        nodePosition += toMousePosition.normalize() *
 availableMovement;
           }
        }

        node->setPosition(nodePosition);

        // Включает/выключает освещение в зависимости от того,
 нажата ли левая кнопка мыши.
        node->setMaterialFlag(video::EMF_LIGHTING,
 receiver.GetMouseState().LeftButtonDown);

        driver->beginScene(true, true,
 video::SColor(255,113,113,133));
        smgr->drawAll(); // отрисовать 3d сцену
        driver->endScene();
 }
        В конце освободить устройство IrrLight.
        device->drop();

        return 0;
 }
```

Задание: создать программу для управления объектом при помощи джойстика.

Тема 6. Трекеры взаимодействия с тренажером

Цель: формирование практических умений при работе с инфракрасным трекером головы.

Теория и примеры выполнения задания

Как известно, инфракрасный трекер – это устройство подобное камере, предназначенное для слежения за положением некоторым образом выделенных точек, связанных с объектом слежения (рис. 4). Отличием инфракрасного датчика от веб-камеры является то, что он может определять расстояние до объекта и реагирует только на интенсивность отраженного сигнала, игнорируя спектр отраженного цвета.

Рис. 4 . Инфракрасный трекер

Для трекинга необходимо не полько получать информацию с датчика, но и обрабатывать получаемые области, вычисляя характеристики объекта слежения. Далее используется устройство TrackIR и отражающие пластины, состоящие из 3 световозвращающих точек.

Для получения информации с датчика нам необходимо подключить соответствующие библиотеки. Этот процесс осуществляется следующим образом:

```
#include "supportcode.h"
#include "cameralibrary.h
#include "modulevector.h"
#include "modulevectorprocessing.h"
#include "coremath.h"
```

В начале работы с трекером головы необходимо инициализировать устройство:

```
int main(int argc, char* argv[])
{
//For OptiTrack Ethernet cameras, it's important to enable
development mode if you
//want to stop execution for an extended time while
debugging without disconnecting
//the Ethernet devices.  Lets do that now:
CameraLibrary_EnableDevelopment();
//Initialize connected cameras
CameraManager::X().WaitForInitialization();
//Get a connected camera
Camera *camera = CameraManager::X().GetCamera();
//If no device connected, pop a message box and exit
if(camera==0)
{
      MessageBox(0,"Please connect a camera","No Device
Connected", MB_OK);
      return 1;
}
//Determine camera resolution to size application window
int cameraWidth  = camera->Width();
int cameraHeight = camera->Height();
int WindowWidth = 800;
int WindowHeight = 450;
//Open the application window
if (!CreateAppWindow("Camera Library SDK - Single Camera
Tracking Sample",WindowWidth,WindowHeight,32,gFullscreen))
return 0;
//Create a texture to push the rasterized camera image
//We're using textures because it's an easy & cpu light
//way to utilize the 3D hardware to display camera
//imagery at high frame rates
```

```
Surface  Texture(cameraWidth, cameraHeight);
Bitmap * framebuffer = new Bitmap(cameraWidth,
cameraHeight, Texture.PixelSpan()*4,
Bitmap::ThirtyTwoBit, Texture.GetBuffer());
//Set Video Mode
camera->SetVideoType(SegmentMode);
//Start camera output
camera->Start();
```

После инициализации мы можем получать данные с трекера в виде областей.

```
while(1)
{
// Fetch a new frame from the camera
Frame *frame = camera->GetFrame();
if(frame)
{
// Ok, we've received a new frame, lets do something
// with it.
// Lets have the Camera Library raster the camera's
// image into our texture.
frame->Rasterize(framebuffer);
vec->BeginFrame();
      for(int i=0; i<frame->ObjectCount(); i++)
{
      cObject *obj = frame->Object(i);

      float x = obj->X();
      float y = obj->Y();
```

Теперь мы можем прорисовать полученные координаты точки.

```
Core::Predistort2DPoint(lensDistortion,x,y);
vec->PushMarkerData(x, y, obj->Area(), obj->Width(),
obj->Height());
```

Полученная двухмерная информация дает представление о том, какие данные получает датчик. По полученным данным можно определить визуально нежелательные блики и исчезнувшие из поля видимости реперные точки (рис. 5). Однако двухмерное представление не позволяет отслеживать трехмерные координаты точек.

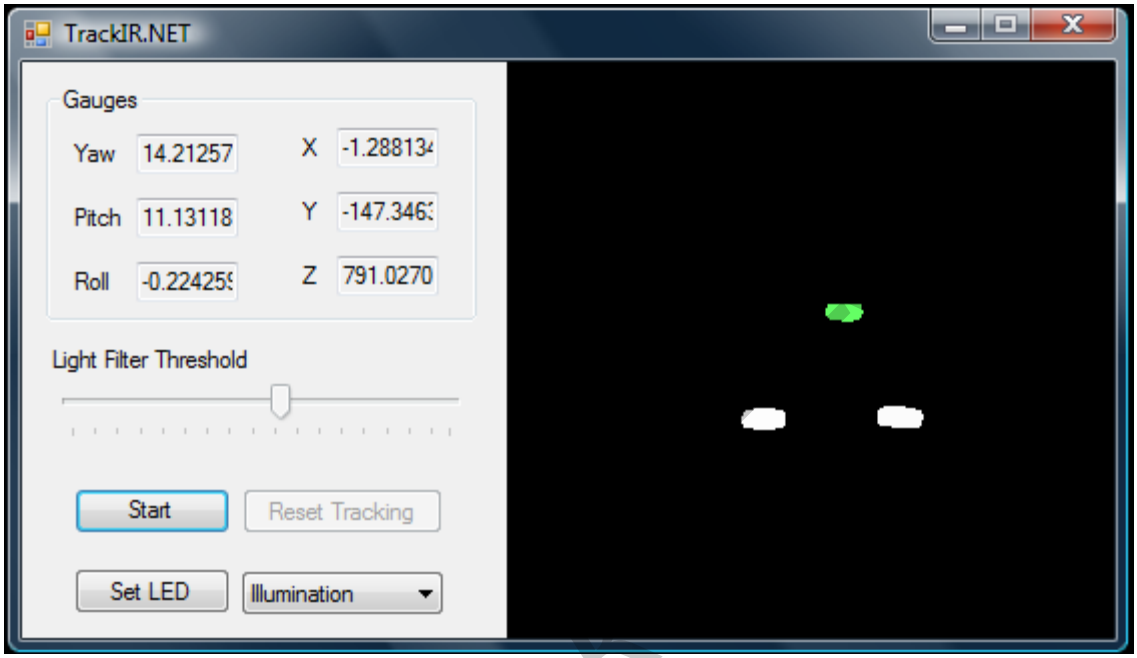

Рис. 5. Определение координат точек

Для получения 3D координат воспользуемся функциями визуального процессора:

```
vec->Calculate();
vecprocessor->PushData(vec);
if(vecprocessor->MarkerCount()==3)
{
      glColor3f(0,1,1);
      glBegin(GL_LINES);
      for(int i=0; I,<vecprocessor->MarkerCount(); i++)
      for(int j=0; j<vecprocessor->MarkerCount(); j++)
{

if(i!=j)
{
```

```
                float x,y,z;
                vecprocessor->GetResult(i,x,y,z);
//рисуем линию в точку
        glVertex3f(x/200,y/200,z/200);
        vecprocessor->GetResult(j,x,y,z);
//рисуем линию в точку
                glVertex3f(x/200,y/200,z/200);
        }
}
        glEnd();
}
```

Далее полученные точки необходимо классифицировать и обработать. Классификация заключается в том, что мы соотносим полученные 3 точки со световозвращающими точками пластины. Обработка заключается в вычислении углов наклона пластины в 3 плоскостях.

Задание: реализовать механизм, управляющий камерой обзора 3D сцены при помощи трекера головы.

Тема 7. Создание эффектов присутствия и взаимодействия

Цель: формирование практических умений создания эффекта присутствия и взаимодействия.

Теория и примеры выполнения задания

Взаимодействие пользователя и виртуальной сцены может быть реализовано при помощи методов определения столкновений (коллизий) между объектами: автоматическое определение при перемещении по 3D пространству (хождение по лестнице, карабканье, скольжение), определение пересечения конкретного полигона модели с лучом, направленным из выбранной камеры, хотя луч можно направить из любой точки пространства.

Как обычно, подключаем заголовочный файл IrrLicht, объявляем пространство имен 'irr'. После выбора драйвера и запуска движка загружается уровень из quake3 и 3 анимированные модели – 4 объекта, на которых тестируется пересечение с лучем, исходящим из камеры.

```
#include < irrlicht.h >
#include "driverChoice.h"
 using namespace irr;
 enum
```

```
    {
    // идентификатор для узлов сцены, которые будут реагировать
    // на вызовы метода getSceneNodeAndCollisionPointFromRay()
    ID_IsNotPickable = 0,

    // идентификатор для узлов сцены, которые не будут
    // проходить проверку на столкновение с лучем
    IDFlag_IsPickable = 1 << 0,

    // идентификатор для узлов сцены, которые будут
    // подсвечиваться при столкновении с лучем
    IDFlag_IsHighlightable = 1 << 1
    };

    int main()
    {
    // запрашиваем у пользователя драйвер (DirectX, OpenGL и
т.д.)
    video::E_DRIVER_TYPE driverType=driverChoiceConsole();
     if (driverType==video::EDT_COUNT)  return 1;

    // создаем корневой объект(движок)
    IrrlichtDevice *device =  createDevice(driverType,
core::dimension2d< u32 >(640, 480), 16, false);

    if (device == 0) return 1; // ошибка при создании движка,
выходим.

    video::IVideoDriver* driver = device->getVideoDriver();
    scene::ISceneManager* smgr = device->getSceneManager();

    device->getFileSystem()-
>addZipFileArchive("../../media/map-20kdm2.pk3");
    scene::IAnimatedMesh* q3levelmesh = smgr-
>getMesh("20kdm2.bsp");
```

```
scene::IMeshSceneNode* q3node = 0;

// Добавляем уровень Quake, как принимающий тесты на
столкновения, указав идентификатор IDFlag_IsPickable
if (q3levelmesh)  q3node = smgr-
>addOctreeSceneNode(q3levelmesh->getMesh(0), 0,
IDFlag_IsPickable);
```

Далее создаем селектор треугольников (triangle selector) – класс, который определяет конкретный полигон в узлах (нодах). Созданный селектор прикрепляется в узле q3node следующим образом:

```
scene::ITriangleSelector* selector = 0;
if (q3node)
{
      q3node->setPosition(core::vector3df(-1350,-130,-
1400));
      selector = smgr->createOctreeTriangleSelector(q3node-
>getMesh(), q3node, 128);
      q3node->setTriangleSelector(selector);
// т.к. селектор нам еще пригодится, мы не будем здесь
выполнять selector->drop()
}
```

Далее к камере подключается аниматор. Как известно, аниматор влияет на узел (нод) сцены таким образом, что добавляет влияние на него гравитации и ощущение препятствий (стен, лестниц и т. д.). Аниматор определяет, с каким типом мира он имеет дело, каким типом нода и как велика гравитация и т. п. Аниматор определяет вероятность столкновений. Он может быть прикреплен не только к камере, но и к любому другому объекту на сцене.

Сейчас подробно рассмотрим параметры метода createCollisionResponseAnimator(). Первый – это селектор треугольников (TriangleSelector). Он занимается поиском треугольников в базе сцены. Второй – узел на сцене, который является инициатором столкновений, в нашем случае это камера, из которой мы будет направлять луч. Третий параметр – радиус объекта инициатора (камеры), к примеру, чем меньше радиус, тем ближе камера сможет подходить к стене. Четвертый параметр определяет силу и направление гравитации, он устанавливается в (0, -10, 0), т. е. направляется в пол с силой 10 г. Значение (0,0,0) отключит гравитацию. Пятый параметр – это

смещение внутри элипсоида, радиус которого мы определили в третьем параметре.

```
    // зададим ускорение прыжка 3 единицы в секунду
    // относительно значения гравитации (0, -10, 0) в аниматоре
определения столкновений.
    scene::ICameraSceneNode* camera = smgr-
>addCameraSceneNodeFPS(0, 100.0f, .3f, ID_IsNotPickable, 0, 0,
true, 3.f);
    camera->setPosition(core::vector3df(50,50,-60));
    camera->setTarget(core::vector3df(-70,30,-60));
    if (selector)
    {
    scene::ISceneNodeAnimator* anim = smgr-
>createCollisionResponseAnimator(
    selector, camera, core::vector3df(30,50,30),
    core::vector3df(0,-10,0), core::vector3df(0,30,0));
    selector->drop(); // вот теперь лично нам селектор больше
не нужен, отключаем его
    camera->addAnimator(anim);
    anim->drop();
    // аниматор столкновений тоже отключаем, оставляя на
попечение камере
    }
    // Теперь добавим персонажей, которых будем «щупать» лучем
камеры, и источник света(билборд),
    // которым будем подсвечивать модель, в которую уперся луч
из камеры
    // спрячем стандартный указатель мыши
    device->getCursorControl()->setVisible(false);
    // Добавляем билборд, который будет подсвечивать место
пересечения луча и объекта, в который он уперся
    scene::IBillboardSceneNode * bill = smgr-
>addBillboardSceneNode();
    bill->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR );
```

```
    bill->setMaterialTexture(0, driver-
>getTexture("../../media/particle.bmp"));
    bill->setMaterialFlag(video::EMF_LIGHTING, false);
    bill->setMaterialFlag(video::EMF_ZBUFFER, false);
    bill->setSize(core::dimension2d< f32 >(20.0f, 20.0f));
    bill->setID(ID_IsNotPickable); // дадим билборду ID для
объектов, которые не реагируют на столкновения.
    Добавляем модельки персонажей и слегка анимируем.
    scene::IAnimatedMeshSceneNode* node = 0;
    // моделька феи в формате MD2 с морф-анимацией
    node = smgr->addAnimatedMeshSceneNode(smgr-
>getMesh("../../media/faerie.md2"), 0, IDFlag_IsPickable |
IDFlag_IsHighlightable);
    node->setPosition(core::vector3df(-70,-15,-120)); //
«поставим» ее на пол
    node->setScale(core::vector3df(2, 2, 2)); // масштабируем
    node->setMD2Animation(scene::EMAT_POINT);
    node->setAnimationSpeed(20.f);
    video::SMaterial material;
    material.setTexture(0, driver-
>getTexture("../../media/faerie2.bmp"));
    material.Lighting = true;
    material.NormalizeNormals = true;
    node->getMaterial(0) = material;
    // создаем селектор и прикрепляем к модельке феи
    selector = smgr->createTriangleSelector(node);
    node->setTriangleSelector(selector);
    selector->drop(); // селектор более не нужен - отключаем.
    // моделька дварфа в формате X со скелетной анимацией, но
без текстур.
    node = smgr->addAnimatedMeshSceneNode(smgr-
>getMesh("../../media/dwarf.x"), 0, IDFlag_IsPickable |
IDFlag_IsHighlightable);
    node->setPosition(core::vector3df(-70,-66,0));
```

```
        // «ставим» дварфа на пол
        node->setRotation(core::vector3df(0,-90,0));
        // поворачиваем лицом к камере
        node->setAnimationSpeed(20.f);
        selector = smgr->createTriangleSelector(node);
        node->setTriangleSelector(selector);
        selector->drop();
        // моделька ниндзи в 3D формате со скелетной анимацией.
        node = smgr->addAnimatedMeshSceneNode(smgr-
>getMesh("../../media/ninja.b3d"), 0, IDFlag_IsPickable |
IDFlag_IsHighlightable);
        node->setScale(core::vector3df(10, 10, 10));
        node->setPosition(core::vector3df(-70,-66,-60));
        node->setRotation(core::vector3df(0,90,0));
        node->setAnimationSpeed(10.f);
        node->getMaterial(0).NormalizeNormals = true;
        // добавляем селектор, как и для предыдущих моделек.
        selector = smgr->createTriangleSelector(node);
        node->setTriangleSelector(selector);
        selector->drop();
        material.setTexture(0, 0);
        material.Lighting = false;
        // добавляем источник света, которым осветим все сразу
модельки персонажей
        scene::ILightSceneNode * light = smgr->addLightSceneNode(0,
core::vector3df(-60,100,400),
video::SColorf(1.0f,1.0f,1.0f,1.0f), 600.0f);
        light->setID(ID_IsNotPickable); // помечаем источник света
как объект, не реагирующий на столкновения.
        // определяем переменную, которая хранит указатель на
подсвеченную модель, и переменную для указателя на менеджер
столкновений
        scene::ISceneNode* highlightedSceneNode = 0;
        scene::ISceneCollisionManager* collMan = smgr-
>getSceneCollisionManager();
```

```
int lastFPS = -1;
// рисуем треугольник, который прошел тест на столкновение, как полый контур(wireframe)
material.Wireframe=true;
while(device->run())
if (device->isWindowActive())
{
driver->beginScene(true, true, 0);
smgr->drawAll();
// включаем реакцию на обычное освещение (указатель *light) для подсвеченного узла(нода), если таковой есть, и «забываем» его(узел)
if (highlightedSceneNode)
{
	highlightedSceneNode->setMaterialFlag(video::EMF_LIGHTING, true);
	highlightedSceneNode = 0;
}
// Все пересечения в примере - это проверка на столкновение с лучем длиной 1000 ед.,
// опущенного из камеры на сцену.  Вы можете легко модифицировать пример под проверку
// траектории пули или положения меча или определение 3D позицей клика указателя мыши
// посредством метода ISceneCollisionManager::getRayFromScreenCoordinates()
core::line3d ray;
ray.start = camera->getPosition();
ray.end = ray.start + (camera->getTarget() - ray.start).normalize() * 1000.0f;
// переменная под хранение точки пересечения
core::vector3df intersection;
// переменная под хранение треугольника, с которым пересекся луч
```

```
core::triangle3df hitTriangle;

// Далее метод, который делает всю работу по определению столкновения луча с объектами сцены.
// Он находит ближайшую точку столкновения и возвращает узел(нод), которому она принадлежит.
// IrrLicht предлагает и другие методы пересечения: луча с треугольником, луча с bounding box,
// Irrlicht provides other types of selection, including ray/triangle selector,
// эллипса с треугольником, плюс несколько методов хелперов(помощников).
// для подробностей смотрите методы класса ISceneCollisionManager
scene::ISceneNode * selectedSceneNode =
        collMan->getSceneNodeAndCollisionPointFromRay(
ray,
intersection,
// точка столкновения
hitTriangle,
// полигон(треугольник), в котором точка столкновения
IDFlag_IsPickable,
// определять столкновения только для нодов с идентификатором IDFlag_IsPickable 0);
// проверять относительно всей сцены (оставляем значение по умолчанию).
// Если луч столкнулся с чем-нибудь, перемещаем указку(билборд) в точку столкновения
// и рисуем треугольник, в котором она находится
if(selectedSceneNode)
{
bill->setPosition(intersection);

// мы должны сбросить трансформации перед отрисовкой.
driver->setTransform(video::ETS_WORLD, core::matrix4());
```

```
        driver->setMaterial(material);
        driver->draw3DTriangle(hitTriangle,
 video::SColor(0,255,0,0));
        // Проверяем идентификатор узла(нода), на который пришлось
 столкновение на тему
        // подсветки. Если ID узла соответствует таковому, то
 запоминаем его и подсвечиваем.
        if((selectedSceneNode->getID() & IDFlag_IsHighlightable) ==
 IDFlag_IsHighlightable)
        {
        highlightedSceneNode = selectedSceneNode;
        // В данном случае мы отключаем обработку освещения(реакцию
 на *light) для узла, на который пришлось
        // столкновение, тогда свет билборда, который подсвечивает
 точку столкновения, разольется по контуру(создаст ореол)
        // (отключаем реакцию на свет, чтобы осветить).
        highlightedSceneNode->setMaterialFlag(video::EMF_LIGHTING,
 false);
        }
}
        // обработка сцены закончена.
        driver->endScene();
        int fps = driver->getFPS();
        if (lastFPS != fps)
{
        core::stringw str = L"Collision detection example -
 Irrlicht Engine [";
        str += driver->getName();
        str += "] FPS:";
        str += fps;
        device->setWindowCaption(str.c_str());
        lastFPS = fps;
        }
}
```

```
device->drop();
return 0;
}
```

Задание: реализовать механизм, препятствующий столкновению камеры с несколькими загруженными на сцену объектами.

Тема 8. Применение компьютерной библиотеки ChronoEngine для имитации физических процессов

Цель: формирование практических умений создания физических процессов при помощи компьютерной библиотеки ChronoEngine.

Теория и примеры выполнения задания

В качестве примера рассмотрим создание простого трехмерного физического эксперимента. Организуем столкновение виртуального массивного шара со стеной, составленной из виртуальных прямоугольных тел (коробок).

Для создания массива графических примитивов, представляющих стену, используется специальный класс ChBodySceneNode. Этот класс является одновременно объектом Irrlicht и элементом ChBody, которые использует библиотека ChronoEngine для расчета физических процессов на виртуальной сцене.

```
void create_some_falling_items(ChSystem& mphysicalSystem,
                ISceneManager* msceneManager, IVideoDriver*
driver) {
ChBodySceneNode* mrigidBody;
video::ITexture* cubeMap = driver->

getTexture("../data/cubetexture.png");
for (int ai = 0; ai < 1; ai++) {
      for (int bi = 0; bi < 10; bi++) {
            for (int ui = 0; ui < 15; ui++) {
      mrigidBody =
            (ChBodySceneNode*) addChBodySceneNode_easyBox(
                  &mphysicalSystem, msceneManager,
                  0.8,
                  ChVector<>(-8+ui*4.0+2*(bi%2),
                                    1.0+bi*2.0, ai*6),
```

```
                    ChQuaternion<>(1,0,0,0),
                    ChVector<>(3.96,2,4) );
            mrigidBody->GetBody()->SetFriction(0.4f);
            mrigidBody->setMaterialTexture(0,     cubeMap);
                  }
            }
      }
```

Для создания площадки, на которой будет выполняться сцена, используется следующий код:

```
            mrigidBody =
 (ChBodySceneNode*)addChBodySceneNode_easyBox(
                  &mphysicalSystem, msceneManager,
                  1.0,
                  ChVector<>(0,-2,0),
                  ChQuaternion<>(1,0,0,0),
                  ChVector<>(550,4,550) );
      mrigidBody->GetBody()->SetBodyFixed(true);
      mrigidBody->GetBody()->SetFriction(0.4f);
```

Добавим шар, который будет сталкиваться с созданной нами стеной:

```
      double mradius = 3;
      double density = 1;
      double mmass = (4./3.)*CH_C_PI*pow(mradius,3)*density;
      double minert = (2./5.)* mmass * pow(mradius,2);

      mrigidBody =
 (ChBodySceneNode*)addChBodySceneNode_easySphere(
      &mphysicalSystem, msceneManager,
      mmass, // mass
      ChVector<>(0, 3, -8), // pos
      mradius, // radius
      20,  // hslices, for rendering
```

```
     15); // vslices, for rendering

     // set moment of inertia (more realisic than default
1,1,1).
     mrigidBody->GetBody()-
>SetInertiaXX(ChVector<>(minert,minert,minert));
     mrigidBody->GetBody()->SetPos_dt(ChVector<>(0,0,16));
     mrigidBody->GetBody()->SetFriction(0.4f);

     // Some aesthetics for 3d view..
     mrigidBody->addShadowVolumeSceneNode();
     //mrigidBody->setMaterialTexture(0,   sphereMap);
     }
```

Наконец создадим главную программу с обычной визуализацией Irrlicht:

```
     int main(int argc, char* argv[])
     {
     DLL_CreateGlobals();

     // Создаем физическую систему ChronoENGINE
     ChSystem mphysicalSystem;

     // Создаем визуализацию Irrlicht (запускаем библиотеку
Irrlicht, связываем систему с пользовательским интерфейсом)
     ChIrrAppInterface application(&mphysicalSystem, L"Bricks
test",core::dimension2d<u32>(800,600),false);
```

// Добавляем камеру, свет и небо:

```
     ChIrrWizard::add_typical_Sky   (application.GetDevice());
     ChIrrWizard::add_typical_Lights(application.GetDevice(),
core::vector3df(30.f, 200.f,  90.f), core::vector3df(30.f, 80.f,
-60.f), 590,  400);
```

```
ChIrrWizard::add_typical_Camera(application.GetDevice(),
core::vector3df(-15,14,-30), core::vector3df(0,5,0));
```

// Добавим физическую систему с помощью CHRONO...
// Создаем все твердотелые объекты.

```
create_some_falling_items(mphysicalSystem,
application.GetSceneManager(), application.GetVideoDriver());
```

Зададим параметр LCP_ITERATIVE_SOR_MULTITHREAD, это поможет использовать многоядерные процессоры. Такой пересчет имеет хорошую скорость/точность характеристики.

```
mphysicalSystem.SetLcpSolverType(ChSystem::LCP_ITERATIVE_SO
R_MULTITHREAD);
```

Если включить алгоритм ожидания, объекты, которые пока остаются в покое, будут иметь внутреннее состояние 'sleep' и не будут пересчитываться, что поможет сохранить ресурсы CPU.

```
mphysicalSystem.SetUseSleeping(false);
```

Возможно увеличить максимальную скорость алгоритма пересчета. Скорость напрямую связана с точностью. Чем меньше количество итераций, тем быстрее будет работать симулятор, но точность снизится.

```
mphysicalSystem.SetMaxPenetrationRecoverySpeed(1.6); //
used by Anitescu stepper only
mphysicalSystem.SetIterLCPmaxItersSpeed(40);
mphysicalSystem.SetIterLCPmaxItersStab(20); // unuseful for
Anitescu, only Tasora uses this
```

Также можно включить «горячий старт», чтобы увеличить точность без сильного увеличения счетчика цикла, однако этот метод не всегда рационален и его следует использовать осторожно.

```
mphysicalSystem.SetIterLCPwarmStarting(true);
```

Выполняем моделирование, используя цикл while():

```
while(application.GetDevice()->run())
{
        application.GetVideoDriver()->beginScene(true, true,
SColor(255,140,161,192));
        application.DrawAll();
        mphysicalSystem.DoStepDynamics( 0.01);
        application.GetVideoDriver()->endScene();
}
return 0;
}
```

Задание: для своей виртуальной сцены добавить взаимодействие между объектами при помощи библиотеки ChronoEngine.

Тема 9. Создание связанных объектов при помощи ChronoEngine

Цель: формирование практических умений связывания объектов сцены при помощи компьютерной библиотеки ChronoEng.

Теория и примеры выполнения задания

В качестве примера рассмотрим процесс создания набора связанных маятников, колеблющихся под действием силового поля. Как известно, крепление маятника – это подвижное соединение.

Вначале создадим функцию, которая будет использоваться для создания потока воздуха от вращающегося вентилятора, действующего на все объекты перед ним. Используется функция «аккумулятор энергии».

```
// контейнер для всех тел
void apply_fan_force (    ChSystem* msystem,
// позиция и вращение для вентилятора
        ChCoordsys<>& fan_csys,
// радиус вентилятора
        double aradius,
// скорость вентилятора
        double aspeed,
// плотность (эвристическое значение)
        double adensity)
{
```

```
 for (unsigned int i=0; i<msystem->Get_bodylist()->size(); i++){
      ChBody* abody = (*msystem->Get_bodylist())[i];
      // Сбрасываем все аккумуляторы силы:
      abody->Empty_forces_accumulators();
      // задаем скорость, силу ветра:
      ChVector<> abs_wind(0,0,0);
      // высчитываем позицию занавески в координатах вентилятора:
      ChVector<> mrelpos =
fan_csys.TrasformParentToLocal(abody->GetPos());
      ChVector<> mrelpos_ondisc = mrelpos; mrelpos_ondisc.z=0;
      // если объекты не за вентилятором
      if (mrelpos.z >0)
            if (mrelpos_ondisc.Length() < aradius)
            {
      //Мы внутри потока ветра
      // ветер направлен перпендикулярно диску вентилятора
      abs_wind =
 fan_csys.TrasformLocalToParent(ChVector<>(0,0,1));
      // скорость ветра от вентилятора постоянна
      abs_wind *= -aspeed;
            }

      // сила пропорциональна силе ветра
      // и дальности от вентилятора
      ChVector<> abs_force = ( abs_wind - abody->GetPos_dt() ) *
 adensity;
      // применяем силу к занавеске
      abody->Accumulate_force(abs_force, abody->GetPos(), false);
      }
}
      int main(int argc, char* argv[])
      {
```

```
      // In CHRONO engine, The DLL_CreateGlobals() -
 DLL_DeleteGlobals(); pair is needed if
      // global functions are needed.
      DLL_CreateGlobals();
      // Create a ChronoENGINE physical system
      ChSystem my_system;
      // Create the Irrlicht visualization (open the Irrlicht
 device,
      // bind a simple user interface, etc. etc.)
      ChIrrAppInterface application(&my_system, L"A simple
 pendulum example",core::dimension2d<u32>(800,600),false);


      // Easy shortcuts to add logo, camera, lights and sky in
 Irrlicht scene:
      ChIrrWizard::add_typical_Logo(application.GetDevice());
      ChIrrWizard::add_typical_Sky(application.GetDevice());
      ChIrrWizard::add_typical_Lights(application.GetDevice());
      ChIrrWizard::add_typical_Camera(application.GetDevice(),
 core::vector3df(0,14,-20));
```

Для создания пяти маятников используем цикл for.

```
      for (int k=0;k<5; k++)
      {
            double z_step =(double)k*2.;

            // .. the truss
            ChBodySceneNode* mrigidBody0 =
 (ChBodySceneNode*)addChBodySceneNode_easyBox(
      ChQuaternion<>(1,0,0,
ChVector<>(5,1,0.5) );
            mrigidBody0->GetBody()->SetBodyFixed(true); // the
 truss does not move!
      mrigidBody0->GetBody()->SetCollide(false);
            ChBodySceneNode* mrigidBody1 =
 (ChBodySceneNode*)addChBodySceneNode_easyBox(
```

```
      ChVector<>(1,6,1) );
            mrigidBody1->GetBody()->SetCollide(false);
            ChBodySceneNode* mrigidBody2 =
 (ChBodySceneNode*)addChBodySceneNode_easyBox(
      ChVector<>(6,1,1) );
            mrigidBody2->GetBody()->SetCollide(false);
            ChBodySceneNode* mrigidBody3 =
 (ChBodySceneNode*)addChBodySceneNode_easyBox(
      ChVector<>(1,6,1) );
            mrigidBody3->GetBody()->SetCollide(false);
```

Далее создаем сочленение типа «пункт на линии», с верхним и нижним пределами по направлению скольжения X для точек подвеса маятника.

```
      ChSharedPtr<ChLinkLockPointLine>  my_link_01(new
 ChLinkLockPointLine);
            my_link_01->Initialize(mrigidBody1->GetBody(),
mrigidBody0->GetBody(),
ChCoordsys<>(ChVector<>(0,0,z_step)));

      my_link_01->GetLimit_X()->Set_active(true);
      my_link_01->GetLimit_X()->Set_max( 1.0);
      my_link_01->GetLimit_X()->Set_min(-1.0);
      my_system.AddLink(my_link_01);

      // .. a spherical joint
      //   (exercise, try also: ChSharedPtr<ChLinkLockHook>
 my_link_12(new ChLinkLockHook);
      ChSharedPtr<ChLinkLockSpherical>  my_link_12(new
 ChLinkLockSpherical);
      my_link_12->Initialize(mrigidBody2->GetBody(),
      mrigidBody1->GetBody(),
      ChCoordsys<>(ChVector<>(0,-6,z_step)));
      my_system.AddLink(my_link_12);

      // .. a spherical joint
```

```
      ChSharedPtr<ChLinkLockSpherical>  my_link_23(new
ChLinkLockSpherical);
      my_link_23->Initialize(mrigidBody3->GetBody(),
      mrigidBody2->GetBody(),
      ChCoordsys<>(ChVector<>(6,-6,z_step)));
            my_system.AddLink(my_link_23);

      }
```

Перед запуском цикла моделирования в реальном времени создадим объект «вентилятор», используя Irrlicht связку для загрузки и движения.

```
      double fan_radius = 5.3;
      IAnimatedMesh*     fanMesh = application.GetSceneManager()-
>getMesh("../data/fan2.obj");
      IAnimatedMeshSceneNode* fanNode =
application.GetSceneManager()->addAnimatedMeshSceneNode
(fanMesh);
      fanNode-
>setScale(irr::core::vector3df((irr::f32)fan_radius,(irr::f32)fa
n_radius,(irr::f32)fan_radius));


      // This will help choosing an integration step which
matches the
      // real-time step of the simulation..
      ChRealtimeStepTimer m_realtime_timer;
      my_system.SetLcpSolverType(ChSystem::LCP_ITERATIVE_PMINRES;
      while(application.GetDevice()->run())
      {
      // Irrlicht must prepare frame to draw
      application.GetVideoDriver()->beginScene(true, true,
SColor(255,140,161,192));
      // Irrlicht application draws all 3D objects and all GUI
items
      application.DrawAll();
```

```
     // Draw also a grid on the horizontal XZ plane
     ChIrrTools::drawGrid(application.GetVideoDriver(), 2, 2,
20,20,
            ChCoordsys<>(ChVector<>(0,-20,0),
Q_from_AngX(CH_C_PI_2) ),
            video::SColor(255, 80,100,100), true);

     // Update the position of the spinning fan (an Irrlicht
     // node, which is here just for aesthetical reasons!)
            ChQuaternion<> my_fan_rotation;
            my_fan_rotation.Q_from_AngY(my_system.GetChTime()*-
0.5);
            ChQuaternion<> my_fan_spin;
            my_fan_spin.Q_from_AngZ(my_system.GetChTime()*4);
            ChCoordsys<> my_fan_coord(
            ChVector<>(12, -6, 0),
            my_fan_rotation);
            ChFrame<> my_fan_framerotation(my_fan_coord);
            ChFrame<> my_fan_framespin(ChCoordsys<>(VNULL,
my_fan_spin));
            ChCoordsys<> my_fan_coordsys = (my_fan_framespin >>
my_fan_framerotation).GetCoord();
            ChIrrTools::alignIrrlichtNodeToChronoCsys(fanNode,
my_fan_coordsys);

     // Apply forces caused by fan & wind if Chrono rigid bodies
are
     // in front of the fan, using a simple tutorial function
(see above):
            apply_fan_force(&my_system, my_fan_coord, fan_radius,
2.2, 0.5);
     // HERE CHRONO INTEGRATION IS PERFORMED: THE
     // TIME OF THE SIMULATION ADVANCES FOR A SINGLE
     // STEP: (2x as fast as real-time - looks better:)
```

```
            my_system.DoStepDynamics( 2*
m_realtime_timer.SuggestSimulationStep(0.01) );
            application.GetVideoDriver()->endScene()
      }

      // Remember this at the end of the program, if you started
      // with DLL_CreateGlobals();
      DLL_DeleteGlobals();

      return 0;
```

Задание: для своей виртуальной сцены добавить силу, действующую на объекты при помощи библиотеки ChronoEngine.

Тема 10. Создание компьютерных тренажеров с использованием Irrlicht и ChronoEngine

Цель: формирование практических умений создания простейших компьютерных тренажеров при помощи компьютерных библиотек Irrlicht и ChronoEngine

Теория и примеры выполнения задания

Рассмотрим процесс создания небольшого мобильного робота с вращающимися колесами. Для выполнения перемещений робота используется клавиатура.

Важно отметить, что в этом примере скорость приводов просто поэтапно меняется пользователем, который может использовать клавиши для уменьшения скорости дискретными шагами. Отчасти эта модель является не вполне реалистичной, более реалистичная реализация должна сглаживать повышение скорости или даже скорости модели привода с наложеннием крутящего момента в ответ на нагрузки.

Сразу после добавления пространства имен объявляются некоторые глобальные переменные:

```
      double STATIC_rot_speed=0;
      double STATIC_x_speed=0;
      double STATIC_z_speed=0;
      double STATIC_wheelfriction=0.6;
```

Далее объявим класс, унаследованный от IEventReceiver, он будет использоваться для управления событиями из пользовательского интерфейса, реализованными в движке Irrlicht. Пользователь может нажать выбранные клавиши для изменения скорости двигателей робота.

```
// Этот класс будет использоваться для управления
// событиями из пользовательского интерфейса
class MyEventReceiver : public IEventReceiver
{
public:
MyEventReceiver(ChIrrAppInterface* myapp)
{
// сохраняем указатель на физическую систему и
//другие объекты, чтобы управлять ими при помощи клавиатуры
      app = myapp;
}
bool OnEvent(const SEvent& event)
{
bool OnEvent(const SEvent& event)
{
{

      // проверяем, была ли нажата клавиша
      if (event.EventType == irr::EET_KEY_INPUT_EVENT
                  && !event.KeyInput.PressedDown){
            switch (event.KeyInput.Key){
            case irr::KEY_KEY_Q:
                  STATIC_x_speed+=1.5;
if (STATIC_x_speed>MAX_XZ_SPEED)
            STATIC_x_speed=MAX_XZ_SPEED;
                  return true;
            case irr::KEY_KEY_W:
                  STATIC_x_speed-=1.5;
if (STATIC_x_speed<-MAX_XZ_SPEED)
            STATIC_x_speed=-MAX_XZ_SPEED;
```

```
                        return true;
                case irr::KEY_KEY_A:
                        STATIC_z_speed+=1.5;
    if (STATIC_z_speed>MAX_XZ_SPEED)
                STATIC_z_speed=MAX_XZ_SPEED;
                        return true;
                case irr::KEY_KEY_Z:
                        STATIC_z_speed-=1.5;
    if (STATIC_z_speed<-MAX_XZ_SPEED)
                STATIC_z_speed=-MAX_XZ_SPEED;
                        return true;
                case irr::KEY_KEY_E:
                        STATIC_rot_speed+=0.05;
     if (STATIC_rot_speed>MAX_ROT_SPEED)
                STATIC_rot_speed=MAX_ROT_SPEED;
                        return true;
                case irr::KEY_KEY_R:
                        STATIC_rot_speed-=0.05;
    if (STATIC_rot_speed<-MAX_ROT_SPEED)
                STATIC_rot_speed=-MAX_ROT_SPEED;
                        return true;
                }
        }

        return false;
    }

    private:
    ChIrrAppInterface* app;
    };
```

Эта небольшая функция создает колеса, сделанные из нескольких твердых тел ChBodySceneNode (центральное колесо и много радиальных роликов, выстроенных вокруг колеса при помощи шарнирных соединений). Функция возвращает указатель на центральное колесо.

```
    ChBodySceneNode* create_mecanum_wheel(ChSystem&
mphysicalSystem,
        ISceneManager* msceneManager,
        IVideoDriver* driver,
        ChVector<> shaft_position,
        ChQuaternion<> shaft_alignment,
        double wheel_radius,
        double wheel_width,
        int n_rollers,
        double roller_angle,
        double roller_midradius,
        double roller_mass,
        double spindle_mass)
    {
    // используется для управления позицией и вращением
    // всех элементов
    ChFrameMoving<> ftot(shaft_position, shaft_alignment);

    ChBodySceneNode*  mCentralWheel =
          (ChBodySceneNode*)addChBodySceneNode_easyCylinder(
          &mphysicalSystem, msceneManager,
          spindle_mass,
          shaft_position,
          shaft_alignment,
          ChVector<>(wheel_radius,wheel_width,wheel_radius));

    mCentralWheel->GetBody()->
          SetInertiaXX(ChVector<>(1.2,1.2,1.2));

    mCentralWheel->GetBody()->SetCollide(false);
    mCentralWheel->addShadowVolumeSceneNode();

    video::ITexture* cylinderMap = driver->
          getTexture("../data/pinkwhite.png");
    mCentralWheel->setMaterialTexture(0,  cylinderMap);
```

```
double half_length_roller = 0.5 * wheel_width *
      1.0/(cos(roller_angle));
double roller_elliptical_rad_Hor  = wheel_radius;
double roller_elliptical_rad_Vert = wheel_radius *
      1.0/(cos(roller_angle));

for (int iroller = 0; iroller < n_rollers; iroller++){
      double pitch = CH_C_2PI *
            ((double)iroller/(double)n_rollers);

// Создаем ролики
ChBodySceneNode* mRoller =
(ChBodySceneNode*)addChBodySceneNode_easyBarrel(
      &mphysicalSystem, msceneManager,
      roller_mass,
      ChVector<>(0,0,0),
      roller_elliptical_rad_Hor,
      roller_elliptical_rad_Vert,
      -half_length_roller, +half_length_roller,
      -(wheel_radius - roller_midradius) );
mRoller->setMaterialTexture(0,  cylinderMap);
mRoller->addShadowVolumeSceneNode();
mRoller->GetBody()->
      SetInertiaXX(ChVector<>(0.05,0.005,0.05));

mRoller->GetBody()->SetCollide(true);
mRoller->GetBody()->SetFriction(STATIC_wheelfriction);
ChFrameMoving<> f1( ChVector<>(0, 0,
      -(wheel_radius - roller_midradius)) ,
      Q_from_AngAxis(roller_angle, ChVector<>(0,0,1) )  );
ChFrameMoving<> f2( ChVector<>(0, 0, 0) ,
      Q_from_AngAxis(pitch, ChVector<>(0,1,0) )  );
ChFrameMoving<> f3 = f1 >> f2 >> ftot;
mRoller->GetBody()->ConcatenatePreTransformation(f3);
```

```
// Сделайте вращательные соединения между роликом и
// центральным колесом
ChFrameMoving<> fr( ChVector<>(0, 0, 0) ,
      Q_from_AngAxis(CH_C_PI/2.0, ChVector<>(1,0,0) )  );
ChFrameMoving<> frabs = fr >> f3;
ChSharedPtr<ChLinkLockRevolute> my_link_roller(
      new ChLinkLockRevolute);
my_link_roller->Initialize( mRoller->GetBody(),
      mCentralWheel->GetBody(),
      frabs.GetCoord());
mphysicalSystem.AddLink(my_link_roller);
}
return mCentralWheel;
}
int main(int argc, char* argv[])
{
// Включаем глабальные функции библиотеки CHRONO engine
DLL_CreateGlobals();
// Создаем физическую систему ChronoENGINE
ChSystem mphysicalSystem;
// Создаем визуализацию Irrlicht
ChIrrAppInterface application(&mphysicalSystem, L"Mecanum
robot simulator",core::dimension2d<u32>(800,600),false);
// создаем текст с информацией
IGUIStaticText* textFPS = application.GetIGUIEnvironment()
      ->addStaticText(
      L"Use keys Q,W, A,Z, E,R to move the robot",
      rect<s32>(150,10,430,40), true);

// Создаем камеру, свет, небо и камеру:
ChIrrWizard::add_typical_Logo(application.GetDevice());
ChIrrWizard::add_typical_Sky(application.GetDevice());
ChIrrWizard::add_typical_Lights(application.GetDevice());
ChIrrWizard::add_typical_Camera(application.GetDevice(),
core::vector3df(0,14,-20));
```

Примечание. Мы должны создать приемник событий при помощи нашего класса MyEventReceiver и указать Irrlicht его использовать.

```
MyEventReceiver receiver(&application);
  // добавляем наш приемник сообщений:
application.SetUserEventReceiver(&receiver);

double platform_radius = 8;
double wheel_radius = 3;
double roller_angle = CH_C_PI/4;

// Создаем тело робота в виде цилиндра
ChBodySceneNode*  mTrussPlatform =
      (ChBodySceneNode*)addChBodySceneNode_easyCylinder(
      &mphysicalSystem, application.GetSceneManager(),
      1,
      ChVector<>(0,0,0),
      QUNIT,
      ChVector<>(platform_radius*1.5, 2,
      platform_radius*1.5) );

mTrussPlatform->GetBody()->
      SetInertiaXX(ChVector<>(1.2,1.2,1.2));
mTrussPlatform->GetBody()->SetCollide(true);
mTrussPlatform->addShadowVolumeSceneNode();
// создаем колеса и подключаем их к платформе
ChFrame<> f0( ChVector<>(0, 0, 0) ,
      Q_from_AngAxis(CH_C_PI/2.0, ChVector<>(1,0,0) )  );
ChFrame<> f1( ChVector<>(0, 0, platform_radius) , QUNIT);
ChFrame<> f2_wA( VNULL ,  Q_from_AngAxis(0*(CH_C_2PI/3.0),
      ChVector<>(0,1,0) )  );
ChFrame<> f2_wB( VNULL ,  Q_from_AngAxis(1*(CH_C_2PI/3.0),
      ChVector<>(0,1,0) )  );
ChFrame<> f2_wC( VNULL ,  Q_from_AngAxis(2*(CH_C_2PI/3.0),
      ChVector<>(0,1,0) )  );
ChFrame<> ftot_wA = f0 >> f1 >> f2_wA;
```

```
ChFrame<> ftot_wB = f0 >> f1 >> f2_wB;
ChFrame<> ftot_wC = f0 >> f1 >> f2_wC;
ChBodySceneNode* spindle_A=create_mecanum_wheel(
      mphysicalSystem,
      application.GetSceneManager(),
      application.GetVideoDriver(),
      ftot_wA.GetCoord().pos,         // позиция колес
      ftot_wA.GetCoord().rot,         // угол колес
      wheel_radius,                   // радиус колес
      2.2,                      // высота
      8,                        // количество роликов
      roller_angle,             // угол роликов
      0.65,               // максимальный радиус роликов
      0.1,                // масса каждого ролика
      0.2);                     // масса шпинделя

ChSharedPtr<ChLinkEngine> my_link_shaftA(new ChLinkEngine);
 my_link_shaftA->Initialize( spindle_A->GetBody(),
       mTrussPlatform->GetBody(),
      (f1 >> f2_wA).GetCoord());
my_link_shaftA->Set_eng_mode(ChLinkEngine::ENG_MODE_SPEED);
my_link_shaftA->Get_spe_funct()->Set_yconst(0.0);
mphysicalSystem.AddLink(my_link_shaftA);

ChBodySceneNode* spindle_B = create_mecanum_wheel(
      mphysicalSystem,
      application.GetSceneManager(),
      application.GetVideoDriver(),
      ftot_wB.GetCoord().pos,         // позиция колес
      ftot_wB.GetCoord().rot,         // угол колес
      wheel_radius,             // радиус колес
      2.2,                      // высота
      8,                        // количество роликов
      roller_angle,             // угол роликов
      0.65,               // максимальный радиус роликов
```

```
        0.1,                // масса каждого ролика
        0.2);                       // масса шпинделя

    ChSharedPtr<ChLinkEngine>  my_link_shaftB(new
ChLinkEngine);
     my_link_shaftB->Initialize( spindle_B->GetBody(),
    mTrussPlatform->GetBody(),
    (f1 >> f2_wB).GetCoord());
    my_link_shaftB->Set_eng_mode(ChLinkEngine::ENG_MODE_SPEED);
    my_link_shaftB->Get_spe_funct()->Set_yconst(0.0);
    mphysicalSystem.AddLink(my_link_shaftB);

    ChBodySceneNode* spindle_C = create_mecanum_wheel(
        mphysicalSystem, application.GetSceneManager(),
        application.GetVideoDriver(),
        ftot_wC.GetCoord().pos,         // позиция колес
        ftot_wC.GetCoord().rot,         // угол колес
        wheel_radius,                   // радиус колес
        2.2,                    // высота
        8,                      // количество роликов
        roller_angle,           // угол роликов
        0.65,               // максимальный радиус роликов
        0.1,                // масса каждого ролика
        0.2);                       // масса шпинделя

    ChSharedPtr<ChLinkEngine> my_link_shaftC(new ChLinkEngine);
        my_link_shaftC->Initialize( spindle_C->GetBody(),
        mTrussPlatform->GetBody(),
        (f1 >> f2_wC).GetCoord());
    my_link_shaftC->Set_eng_mode(ChLinkEngine::ENG_MODE_SPEED);
    my_link_shaftC->Get_spe_funct()->Set_yconst(0.0);
    mphysicalSystem.AddLink(my_link_shaftC);

    // Создаем группу соударений
    ChBodySceneNode* ground =
        (ChBodySceneNode*)addChBodySceneNode_easyBox(
```

```
        &mphysicalSystem, application.GetSceneManager(),
        100.0,
        ChVector<>(0,-5,0),
        ChQuaternion<>(1,0,0,0),
        ChVector<>(80,1,80) );
ground->GetBody()->SetBodyFixed(true);
ground->GetBody()->SetFriction(STATIC_wheelfriction);

video::ITexture* cubeMap = application.GetVideoDriver()->
        getTexture("../data/cubetexture.png");
ground->setMaterialTexture(0,   cubeMap);

// Подготавливаем физическую систему к моделированию
mphysicalSystem.SetIntegrationType(ChSystem::INT_TASORA);
mphysicalSystem.SetLcpSolverType(ChSystem::LCP_ITERATIVE_SOR);
mphysicalSystem.SetIterLCPmaxItersSpeed(30);
mphysicalSystem.SetIterLCPmaxItersStab(30);

//Цикл реального времени
while(application.GetDevice()->run()){
        application.GetVideoDriver()->beginScene(true, true,
                SColor(255,140,161,192));
        application.DrawAll();
        // один шаг симулирования
        mphysicalSystem.DoStepDynamics( 0.01);
        // изменяем скорость моторов
        ChVector<> imposed_speed(STATIC_x_speed, 0,
                STATIC_z_speed);
        ChFrame<> roll_twist (ChVector<>(0,-wheel_radius,0),
                Q_from_AngAxis(-roller_angle,
                ChVector<>(0,1,0)) );

        ChFrame<> abs_roll_wA = roll_twist >> f2_wA >>
        ChFrame<>(mTrussPlatform->GetBody()->GetCoord());
        double wheel_A_rotspeed = (STATIC_rot_speed *
                platform_radius) + ((abs_roll_wA.GetA()->
```

```
                    MatrT_x_Vect(imposed_speed)).x /
                    sin(roller_angle))/wheel_radius;
        ChFrame<> abs_roll_wB = roll_twist >> f2_wB >>
        ChFrame<>(mTrussPlatform->GetBody()->GetCoord());
        double wheel_B_rotspeed = (STATIC_rot_speed *
              platform_radius) + ((abs_roll_wB.GetA()->
                    MatrT_x_Vect(imposed_speed)).x /
                    sin(roller_angle))/wheel_radius;
        ChFrame<> abs_roll_wC = roll_twist >> f2_wC >>
        ChFrame<>(mTrussPlatform->GetBody()->GetCoord());
        double wheel_C_rotspeed = (STATIC_rot_speed *
              platform_radius) + ((abs_roll_wC.GetA()->
                    MatrT_x_Vect(imposed_speed)).x /
                    sin(roller_angle))/wheel_radius;

        my_link_shaftA->Get_spe_funct()->
              Set_yconst(wheel_A_rotspeed);
        my_link_shaftB->Get_spe_funct()->
              Set_yconst(wheel_B_rotspeed);
        my_link_shaftC->Get_spe_funct()->
              Set_yconst(wheel_C_rotspeed);

        application.GetVideoDriver()->endScene();
    }
    // Отключаем использование глобальных функций
    DLL_DeleteGlobals();

    return 0;
    }
```

Задание: создать колесную базу робота и систему управление ее движением при помощи клавиатуры.

Тема 11. Компьютерная библиотека интеллектуальной обработки изображений OpenCV

Цель: формирование практических умений при работе с компьютерной библиотекой интеллектуальной обработки изображений OpenCV на примере интеллектуальной обработки цветовой гаммы изображений.

Теория и примеры выполнения задания

Библиотека OpenCV предназначена для работы с графической информацией и содержит большое количество хорошо оптимизированных функций для управления, обработки и анализа изображений.

Как известно, изображение – это массив пикселей.

Для реализации изображения используется функция cvThreshold()

```
CVAPI(double)  cvThreshold( const CvArr*  src, CvArr*  dst,
       double  threshold, double  max_value,
       int threshold_type );
```

Эта функция выполняет:

1) фиксированное пороговое преобразование для элементов массива:

src – исходный массив (изображение) (одноканальное, 8-битное или 32-битное);

dst – целевой массив (должен иметь тот же тип, что и src или 8-битный);

threshold – пороговая величина;

max_value – максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV)

threshold_type – тип порогового преобразования:

```
#define CV_THRESH_BINARY      0
#define CV_THRESH_BINARY_INV  1
#define CV_THRESH_TRUNC       2
#define CV_THRESH_TOZERO      3
#define CV_THRESH_TOZERO_INV  4
#define CV_THRESH_MASK        7
#define CV_THRESH_OTSU        8
```

CV_THRESH_BINARY

$$\texttt{dst}(x,y) = \begin{cases} \texttt{max\_value} & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ 0 & \text{otherwise} \end{cases}$$

CV_THRESH_BINARY_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ \texttt{max\_value} & \text{otherwise} \end{cases}$$

CV_THRESH_TRUNC

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

CV_THRESH_TOZERO

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ 0 & \text{otherwise} \end{cases}$$

CV_THRESH_TOZERO_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

```
CVAPI(void)  cvAdaptiveThreshold( const CvArr* src,
       CvArr* dst, double max_value,
       int adaptive_method CV_DEFAULT(CV_ADAPTIVE_THRESH_MEAN_C),
       int threshold_type CV_DEFAULT(CV_THRESH_BINARY),
       int block_size CV_DEFAULT(3),
       double param1 CV_DEFAULT(5));
```

2) адаптивное пороговое преобразование для элементов массива:

src – исходное изображение;
dst – целевое изображение;
max_value – максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV);
adaptive_method – используемый адаптационный алгоритм:

```
#define              CV_ADAPTIVE_THRESH_MEAN_C              0
#define CV_ADAPTIVE_THRESH_GAUSSIAN_C  1
```

threshold_type – тип порогового преобразования;
block_size – размер окрестности (в пикселях), которая используется для расчета порогового значения: 3, 5, 7 и т. д.
param1 – параметр, зависящий от используемого метода. Для методов CV_ADAPTIVE_THRESH_MEAN_C
CV_ADAPTIVE_THRESH_GAUSSIAN_C – это константа, вычитаемая из среднего или взвешенного значения (может быть отрицательной).

Пример порогового преобразования:

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
        IplImage *src=0, *dst=0, *dst2=0;
        // имя картинки задается первым параметром
        char* filename = argc >= 2 ? argv[1] :
"Image0.jpg";
        // получаем картинку
        src = cvLoadImage(filename, 0);
        printf("[i] image: %s\n", filename);
        assert( src != 0 );
        // покажем изображение
        cvNamedWindow( "original", 1 );
        cvShowImage( "original", src );
        dst = cvCreateImage( cvSize(src->width, src-
>height),
             IPL_DEPTH_8U, 1);
        dst2 = cvCreateImage( cvSize(src->width, src-
>height),
             IPL_DEPTH_8U, 1);
        cvThreshold(src, dst, 50, 250, CV_THRESH_BINARY);
        cvAdaptiveThreshold(src, dst2, 250,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 7, 1);
        // показываем результаты
        cvNamedWindow( "cvThreshold", 1 );
        cvShowImage( "cvThreshold", dst);
        cvNamedWindow( "cvAdaptiveThreshold", 1 );
        cvShowImage( "cvAdaptiveThreshold", dst2);
        // ждем нажатия клавиши
        cvWaitKey(0);
        // освобождаем ресурсы
        cvReleaseImage(&src);
        cvReleaseImage(&dst);
        cvReleaseImage(&dst2);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

Аналогичный результат можно получить, используя функции для выборки тех пикселей изображения, которые лежат в заданном интервале значений (могут храниться как в массиве cvInRange(), так и задаваться скалярами cvInRangeS()):

```
CVAPI(void) cvInRange( const CvArr* src, const CvArr* lower,
                       const CvArr* upper, CvArr* dst );
```

Поэлементная проверка массива удостоверяет:

1) что значения элементов массива лежат между значениями элементов двух других массивов:

dst(idx) = lower(idx) <= src(idx) < upper(idx)

src – исходный массив;
lower – массив с нижней границей (включая);
upper – массив с верхней границей (не включая);
dst – массив для хранения результата (тип 8S или 8U)

```
CVAPI(void) cvInRangeS( const CvArr* src, CvScalar lower,
                        CvScalar upper, CvArr* dst );
```

2) что элемент массива лежит между двух скаляров по формуле:

dst(idx) = lower <= src(idx) < upper

src – исходный массив;
lower – скаляр с нижней границей (включая);
upper – скаляр с верхней границей (не включая);
dst – массив для хранения результата (тип 8S или 8U).

В качестве примера рассмотрим выборку определенных областей изображения, которые отличаются от других яркостью (в случае изображения одноканального изображения (в градациях серого)) или цветом (в случае цветного изображения). На вход cvThreshold должно поступать одноканальное, т. к. пороговое преобразование работает с яркостью.

Можно обработать и цветное RGB изображение: для этого его сначала придется разбить на слои (cvSplit()), проделать пороговое преобразование над каждым слоем, сложить слои вместе (cvMerge()). Дополним пример с RGB слоями:

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
        IplImage *image=0, *gray=0, *dst=0, *dst2=0;
        IplImage* r=0, *g=0, *b=0; // для хранения отдельных слоев
RGB изображения
        // имя картинки задается первым параметром
```

```
        char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
        // получаем картинку
        image = cvLoadImage(filename, 1);

        printf("[i] image: %s\n", filename);
        assert( image != 0 );

        // покажем изображение
        cvNamedWindow( "original", 1 );
        cvShowImage( "original", image );

        // картинка для хранения изображения в градациях серого
        gray = cvCreateImage(cvGetSize(image), image->depth, 1);

        // преобразуем картинку в градациях серого
        cvConvertImage(image, gray, CV_BGR2GRAY);

        // покажем серую картинку
        cvNamedWindow( "gray", 1 );
        cvShowImage( "gray", gray );

        dst = cvCreateImage( cvGetSize(gray), IPL_DEPTH_8U, 1);
        dst2 = cvCreateImage( cvGetSize(gray), IPL_DEPTH_8U, 1);

        // пороговые преобразования над картинкой в градациях
серого
        cvThreshold(gray, dst, 50, 250, CV_THRESH_BINARY);
        cvAdaptiveThreshold(gray, dst2, 250,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 7, 1);

        // показываем результаты
        cvNamedWindow( "cvThreshold", 1 );
        cvShowImage( "cvThreshold", dst);
        cvNamedWindow( "cvAdaptiveThreshold", 1 );
        cvShowImage( "cvAdaptiveThreshold", dst2);

        // проведем пороговое преобразование над RGB картинкой

        r = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
        g = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
        b = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);

        // разбиваем на отдельные слои
        cvSplit(image, b, g, r, 0);

        // картинка для хранения промежуточных результатов
        IplImage* temp = cvCreateImage( cvGetSize(image),
IPL_DEPTH_8U, 1 );

        // складываем картинки с одинаковым весом
        cvAddWeighted( r, 1.0/3.0, g, 1.0/3.0, 0.0, temp );
```

```
        cvAddWeighted( temp, 2.0/3.0, b, 1.0/3.0, 0.0, temp );

        // выполняем пороговое преобразование
        cvThreshold(temp, dst, 50, 250, CV_THRESH_BINARY);

        cvReleaseImage(&temp);

        // показываем результат
        cvNamedWindow( "RGB cvThreshold", 1 );
        cvShowImage( "RGB cvThreshold", dst);

        //
        // попробуем пороговое преобразование над отдельными слоями
        //

        IplImage* t1, *t2, *t3; // для промежуточного хранения
        t1 = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
        t2 = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);
        t3 = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1);

        // выполняем пороговое преобразование
        cvThreshold(r, t1, 50, 250, CV_THRESH_BINARY);
        cvThreshold(g, t2, 50, 250, CV_THRESH_BINARY);
        cvThreshold(b, t3, 50, 250, CV_THRESH_BINARY);

        // складываем результаты
        cvMerge(t3, t2, t1, 0, image);
        cvNamedWindow( "RGB cvThreshold 2", 1 );
        cvShowImage( "RGB cvThreshold 2", image);

        cvReleaseImage(&t1); cvReleaseImage(&t2);
cvReleaseImage(&t3);

        // ждем нажатия клавиши
        cvWaitKey(0);

        // освобождаем ресурсы
        cvReleaseImage(& image);
        cvReleaseImage(&gray);
        cvReleaseImage(&dst);
        cvReleaseImage(&dst2);
        cvReleaseImage(&r); cvReleaseImage(&g);
cvReleaseImage(&b);

        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

Задание: создать программу для порогового преобразования изображения.

Тема 12. Поиск объектов на основе цветовой гаммы с использованием OpenCV

Цель: формирование практических умений при работе с алгоритмами поиска объектов на фотографии на основе цветовой гаммы при помощи компьютерной библиотеки интеллектуальной обработки изображений OpenCV.

Теория и примеры выполнения задания

Как известно, самый распространенный способ выделить объект – это цвет. Цвет – свойство тел отражать или испускать видимое излучение определенного спектрального состава и интенсивности.

При поиске по цвету имеет влияние множество факторов, например освещенность. Нельзя также забывать, что видимый цвет – это результат взаимодействия спектра излучаемого света и поверхности. То есть если белый лист освещать светом красной лампы, то и лист будет казаться красным.

Трихроматическая теория (сетчатка глаза имеет 3 вида рецепторов света, ответственных за цветное зрение) полагает, что достаточно всего 3 числа, чтобы описать цвет (красный, синий, зеленый), т. е. используя 3 значения R, G, B. Как известно, цветовые пространства бывают линейные и нелинейные: к линейным относится RGB, нелинейные – HSV, LAB.

Удобство HSV состоит в том, что координаты выбраны с учетом человеческого восприятия:

– Hue (Тон);

– Saturation (Насыщенность);

– Value (Intensity) (Интенсивность).

Рассмотрим практическую задачу – разработку системы машинного зрения: над колесом игорной рулетки установлена камера, требуется создать программу распознавания номера, в который попал шарик.

Исходные данные:

– шарик лежит во внешнем углу ячейки;

– номера на всех колесах располагаются одинаково;

– цвет номеров строго чередуется;

– номер 0 (Zero) зеленого цвета.

Один из возможных алгоритмов решения задачи – найти зеленый сектор Zero и отсчитать от него сектор, в котором детектируется белый шарик. Таким образом, в первом приближении задача состоит в нахождении цветных объектов на картинке.

Создадим алгоритм, который считывает картинку (в качестве первого параметра), разбивает ее на слои (cvSplit()), над каждым из которых можно проделать пороговое преобразование (cvInRangeS()), причем значения интервалов удобным образом изменяются с помощью ползунков.

Результирующее значение выводится в окне «rgb and» и представляет собой логическое И – cvAnd() между получившимися пороговыми картинками.

```
#include <cv.h>
```

```
    #include <highgui.h>
    #include <stdlib.h>
    #include <stdio.h>

    IplImage* image = 0;
    IplImage* dst = 0;

    // для хранения каналов RGB
    IplImage* rgb = 0;
    IplImage* r_plane = 0;
    IplImage* g_plane = 0;
    IplImage* b_plane = 0;
    // для хранения каналов RGB после преобразования
    IplImage* r_range = 0;
    IplImage* g_range = 0;
    IplImage* b_range = 0;
    // для хранения суммарной картинки
    IplImage* rgb_and = 0;

    int Rmin = 0;
    int Rmax = 256;

    int Gmin = 0;
    int Gmax = 256;

    int Bmin = 0;
    int Bmax = 256;

    int RGBmax = 256;

    //
    // функции-обработчики ползунка
//
void myTrackbarRmin(int pos) {
       Rmin = pos;
       cvInRangeS(r_plane, cvScalar(Rmin), cvScalar(Rmax),
r_range);
}

void myTrackbarRmax(int pos) {
       Rmax = pos;
       cvInRangeS(r_plane, cvScalar(Rmin), cvScalar(Rmax),
r_range);
}

void myTrackbarGmin(int pos) {
       Gmin = pos;
       cvInRangeS(g_plane, cvScalar(Gmin), cvScalar(Gmax),
g_range);
}

void myTrackbarGmax(int pos) {
```

```
        Gmax = pos;
        cvInRangeS(g_plane, cvScalar(Gmin), cvScalar(Gmax),
g_range);
}

void myTrackbarBmin(int pos) {
        Bmin = pos;
        cvInRangeS(b_plane, cvScalar(Bmin), cvScalar(Bmax),
b_range);
}

void myTrackbarBmax(int pos) {
        Bmax = pos;
        cvInRangeS(b_plane, cvScalar(Bmin), cvScalar(Bmax),
b_range);
}

int main(int argc, char* argv[])
{
        // имя картинки задается первым параметром
        char* filename = argc == 2 ? argv[1] : "Image0.jpg";
        // получаем картинку
        image = cvLoadImage(filename,1);

        printf("[i] image: %s\n", filename);
        assert( image != 0 );

        // создаем картинки
        rgb = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 3 );
    r_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    g_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    b_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    r_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    g_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    b_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    rgb_and = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
    //  копируем
    cvCopyImage(image, rgb);
    // разбиваем на отельные каналы
    cvSplit( rgb, b_plane, g_plane, r_plane, 0 );

    //
    // определяем минимальное и максимальное значение
    // у каналов HSV
    double framemin=0;
    double framemax=0;

    cvMinMaxLoc(r_plane, &framemin, &framemax);
    printf("[R] %f x %f\n", framemin, framemax );
    Rmin = framemin;
    Rmax = framemax;
```

```
    cvMinMaxLoc(g_plane, &framemin, &framemax);
    printf("[G] %f x %f\n", framemin, framemax );
    Gmin = framemin;
    Gmax = framemax;
    cvMinMaxLoc(b_plane, &framemin, &framemax);
    printf("[B] %f x %f\n", framemin, framemax );
    Bmin = framemin;
    Bmax = framemax;

    // окна для отображения картинки
    cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("R",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("G",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("B",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("R range",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("G range",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("B range",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("rgb and",CV_WINDOW_AUTOSIZE);

    cvCreateTrackbar("Rmin", "R range", &Rmin, RGBmax,
myTrackbarRmin);
    cvCreateTrackbar("Rmax", "R range", &Rmax, RGBmax,
myTrackbarRmax);
    cvCreateTrackbar("Gmin", "G range", &Gmin, RGBmax,
myTrackbarGmin);
    cvCreateTrackbar("Gmax", "G range", &Gmax, RGBmax,
myTrackbarGmax);
    cvCreateTrackbar("Bmin", "B range", &Gmin, RGBmax,
myTrackbarBmin);
    cvCreateTrackbar("Bmax", "B range", &Gmax, RGBmax,
myTrackbarBmax);

//
// разместим окна на рабочем столе
//
 if(image->width <1920/4 && image->height<1080/2){
              cvMoveWindow("original", 0, 0);
              cvMoveWindow("R", image->width+10, 0);
              cvMoveWindow("G", (image->width+10)*2, 0);
              cvMoveWindow("B", (image->width+10)*3, 0);
              cvMoveWindow("rgb and", 0, image->height+30);
              cvMoveWindow("R range", image->width+10, image-
>height+30);
              cvMoveWindow("G range", (image->width+10)*2,
image->height+30);
              cvMoveWindow("B range", (image->width+10)*3,
image->height+30);
       }

       while(true){

              // показываем картинку
```

```
            cvShowImage("original",image);

            // показываем слои
            cvShowImage( "R", r_plane );
            cvShowImage( "G", g_plane );
            cvShowImage( "B", b_plane );

            // показываем результат порогового преобразования
            cvShowImage( "R range", r_range );
            cvShowImage( "G range", g_range );
            cvShowImage( "B range", b_range );

            // складываем
            cvAnd(r_range, g_range, rgb_and);
            cvAnd(rgb_and, b_range, rgb_and);

            // показываем результат
            cvShowImage( "rgb and", rgb_and );

            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC - выходим
                    break;
            }
        }
        printf("\n[i] Results:\n" );
        printf("[i][R] %d : %d\n", Rmin, Rmax );
        printf("[i][G] %d : %d\n", Gmin, Gmax );
        printf("[i][B] %d : %d\n", Bmin, Bmax );

        // освобождаем ресурсы
        cvReleaseImage(& image);
        cvReleaseImage(&rgb);
        cvReleaseImage(&r_plane);
        cvReleaseImage(&g_plane);
        cvReleaseImage(&b_plane);
        cvReleaseImage(&r_range);
        cvReleaseImage(&g_range);
        cvReleaseImage(&b_range);
        cvReleaseImage(&rgb_and);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

У вас могут получиться следующие значения:

```
R [0, 61)
G [135, 255)
B [47, 255)
```

Итак, сектор Zero выделяется хорошо, но попытка выделить по цвету шарик не даст ожидаемого результата. Разумеется, можно добиться, чтобы шарик был виден, но также будут видны белые числа номеров, а еще блики (на оси рулетки и металлической ленте).

Задание: создать программу для определения выпавшего числа на рулетке.

Тема 13. Нахождение границ рисованных объектов

Цель: формирование практических умений при работе с компьютерной библиотекой интеллектуальной обработки изображений OpenCV на примере нахождения границ нарисованных объектов.

Теория и примеры выполнения задания

Как известно, края (границы) – это такие кривые на изображении, вдоль которых происходит резкое изменение яркости или других видов неоднородностей. Проще говоря, край – это резкий переход/изменение яркости.

Причины возникновения краев:

- изменение освещенности;
- изменение цвета;
- изменение глубины сцены (ориентации поверхности).

Края отражают важные особенности изображения, и поэтому целями преобразования изображения в набор кривых являются:

- выделение существенных характеристик изображения;
- сокращение объема информации для последующего анализа.

Самым популярным методом выделения границ является детектор границ Кенни.

Шаги детектора:

– убрать шум и лишние детали из изображения;

– рассчитать градиент изображения;

– сделать края тонкими (edge thinning);

– связать края в контур (edge linking).

Детектор использует фильтр на основе первой производной от гауссианы. Так как он восприимчив к шумам, то лучше не применять данный метод на необработанных изображениях. Сначала исходные изображения нужно свернуть с гауссовым фильтром.

Границы на изображении могут находиться в различных направлениях, поэтому алгоритм Кенни использует четыре фильтра для выявления горизонтальных, вертикальных и диагональных границ. При использовании оператора обнаружения границ (например оператора Собеля) получается значение для первой производной в горизонтальном (Gy) и вертикальном направлении (Gx).

Из этого градиента можно получить угол направления границы:

$$Q = \arctan(Gx/Gy).$$

Угол направления границы округляется до одного из четырех углов, представляющих вертикаль, горизонталь и две диагонали (например, 0, 45, 90 и 135°). Затем проверяется, достигает ли величина градиента локального максимума в соответствующем направлении.

Например, для сетки 3×3:

- если угол направления градиента равен нулю, точка будет считаться границей, если ее интенсивность больше, чем у точек, расположенных выше и ниже рассматриваемой точки;
- если угол направления градиента равен 90°, точка будет считаться границей, если ее интенсивность больше, чем у точкек, расположенных слева и справа от рассматриваемой точки;
- если угол направления градиента равен 135°, точка будет считаться границей, если ее интенсивность больше, чем у точек, находящихся в верхнем левом и нижнем правом углу от рассматриваемой точки;
- если угол направления градиента равен 45°, точка будет считаться границей, если ее интенсивность больше, чем у точек, находящихся в верхнем правом и нижнем левом углу от рассматриваемой точки.

Таким образом, получается двоичное изображение, содержащее границы (так называемые «тонкие края»). В OpenCV детектор границ Кенни реализуется функцией cvCanny(), которая обрабатывает только одноканальные изображения.

```
CVAPI(void)  cvCanny( const CvArr* image, CvArr* edges, double threshold1,
                      double threshold2, int  aperture_size CV_DEFAULT(3) );
```

Выполнение алгоритма Canny для поиска границ:

image – одноканальное изображение для обработки (градации серого);

edges – одноканальное изображение для хранения границ, найденных функцией;

threshold1 – порог минимума;

threshold2 – порог максимума;

aperture_size – размер для оператора Собеля.

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* gray = 0;
```

```
IplImage* dst = 0;

int main(int argc, char* argv[])
{
        // имя картинки задается первым параметром
        char* filename = argc == 2 ? argv[1] : "Image0.jpg";
        // получаем картинку
        image = cvLoadImage(filename,1);

        printf("[i] image: %s\n", filename);
        assert( image != 0 );

        // создаем одноканальные картинки
        gray = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
        dst = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );

        // окно для отображения картинки
        cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
        cvNamedWindow("gray",CV_WINDOW_AUTOSIZE);
        cvNamedWindow("cvCanny",CV_WINDOW_AUTOSIZE);
        // преобразуем в градации серого
        cvCvtColor(image, gray, CV_RGB2GRAY);
        // получаем границы
        cvCanny(gray, dst, 10, 100, 3);
        // показываем картинки
        cvShowImage("original",image);
        cvShowImage("gray",gray);
        cvShowImage("cvCanny", dst );
        // ждем нажатия клавиши
        cvWaitKey(0);
        // освобождаем ресурсы
        cvReleaseImage(&image);
        cvReleaseImage(&gray);
        cvReleaseImage(&dst);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

Обратите внимание, как меняется картина, если увеличить размер оператора Собеля. Вот результат работы функции:

cvCanny(gray, dst, 10, 100, 5);

Полученные границы можно накладывать на сам объект, добиваясь эффекта ретуширования фотографии.

Задание: создать программу для ретуширования фотографий.

Тема 14. Обработка потокового видео с использованием изображений OpenCV

Цель: формирование практических умений при работе с потоковым видео при помощи компьютерной библиотеки интеллектуальной обработки изображений OpenCV.

Теория и примеры выполнения задания

Как известно, работа с камерой почти ничем не отличается от работы с видео, кроме того, что вместо функции cvCreateFileCapture() нужно использовать функцию cvCreateCameraCapture(), которая в качестве параметра принимает не название файла, а идентификатор камеры.

Данная программа представляет собой удобную утилиту для работы с камерой. Программа подключается и начинает захват с камеры с помощью cvCreateCameraCapture(), далее получает ширину и высоту кадра с помощью cvGetCaptureProperty(), а потом в цикле при помощи cvQueryFrame() получает изображение с камеры и выводит в окно.

При нажатии клавиши ESC программа выйдет из цикла и завершится, а при нажатии клавиши Enter текущий кадр будет сохранен в файл ImageN.jpg, где N – номер кадра, начиная от 0 (Image0.jpg, Image1.jpg и т. д.).

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
// получаем любую подключенную камеру
CvCapture* capture = cvCreateCameraCapture(CV_CAP_ANY);
//cvCaptureFromCAM( 0 );
assert( capture );

//cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH,
640);//1280);
//cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT,
480);//960);

// узнаем ширину и высоту кадра
     double width = cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH);
      double height = cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT);
printf("[i] %.0f x %.0f\n", width, height );

IplImage* frame=0;

cvNamedWindow("capture", CV_WINDOW_AUTOSIZE);
```

```
printf("[i] press Enter for capture image and Esc for quit!\n\n");

int counter=0;
char filename[512];

while(true){
// получаем кадр
      frame = cvQueryFrame( capture );

// показываем
cvShowImage("capture", frame);

char c = cvWaitKey(33);
if (c == 27) { // нажата ESC
break;
}
else if(c == 13) { // Enter
// сохраняем кадр в файл
sprintf(filename, "Image%d.jpg", counter);
printf("[i] capture... %s\n", filename);
cvSaveImage(filename, frame);
counter++;
      }
}
// освобождаем ресурсы
cvReleaseCapture( &capture );
cvDestroyWindow("capture");
return 0;
}
```

Основная функция начинает захват с камеры:

```
#define cvCaptureFromCAM cvCreateCameraCapture

CVAPI(CvCapture*) cvCreateCameraCapture( int index );
```

index – номер камеры в системе (состоит из суммы порядкового номера и так называемого домена),

0 – первая попавшаяся камера (можно использовать, если работает всего одна камера).

Задание: создать программу для захвата видео из камеры.

Тема 15. Алгоритмы OpenCV трекинга глаз, головы и рук

Цель: формирование практических умений по организации трекинга глаз, головы и рук в компьютерной библиотеке OpenCV.

Теория и примеры выполнения задания

Рассмотрим алгоритм, который стоит на стыке таких областей, как Machine Learning и Computer Vision.

Представленная методика впервые была описана в статье «Rapid Object Detection using a Boosted Cascade of Simple Features» (Paul Viola, Michael Jones, 2001). С тех пор она получила признание и широкое распространение в области поиска объектов на изображениях или в видеопотоке.

Описываемая методика применялась в области разработки алгоритмов для детекта лиц, но впоследствии стала использоваться и для поиска объектов на изображениях.

В основе методики лежит алгоритм adaptive boosting'a (адаптивного усиления) или сокращенно AdaBoost. Смысл алгоритма заключается в том, что если у нас есть набор эталонных объектов, т. е. есть значения и класс, к которому они принадлежат (например, –1 – нет лица, +1 – есть лицо), кроме того, имеется множество простых классификаторов, то мы можем составить один более совершенный и мощный классификатор. При этом в процессе составления или обучения финального классификатора акцент делается на эталоны, которые распознаются «хуже», в этом и заключается адаптивность алгоритма, в процессе обучения он подстраивается под наиболее «сложные» объекты.

На рис. 6 вы видите набор примитивов. Чтобы понять суть методики, представьте, что мы берем эталонное изображение и накладываем на него какой-либо из примитивов, например 1a, далее считаем сумму значений пикселей в белой области примитива (левая часть) и черной области (правая часть) и отнимаем от первого значения второе. В итоге получаем обобщенную характеристику анизотропии некоторого участка изображения.

Но тут возникает проблема. Даже для небольшого изображения количество накладываемых примитивов очень велико, если взять изображение размером 24×24, то количество примитивов ~180 000. Задача алгоритма AdaBoost выбрать те примитивы, которые наиболее эффективно выделяют данный объект.

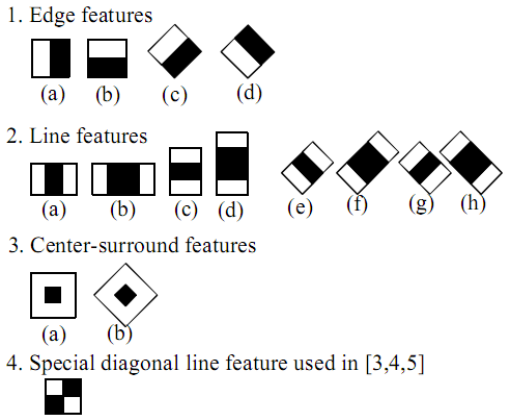

Рис. 6. Набор примитивов

Для объекта слева алгоритм выбрал два примитива. По понятным причинам область глаз более темная по сравнению со средней областью лица и переносицы. Примитивы этой конфигурации и размеров наиболее лучшим образом «характеризуют» данное изображение.

На основе таких классификаторов с отобранными наиболее эффективными примитивами строится каскад. Каждый последующий элемент каскада имеет более жесткие условия успешного прохождения, чем предыдущий (используется больше примитивов), тем самым до конца доходят только самые «правильные».

Алгоритм обучения очень длительный процесс. При необходимом подходе он может длиться 3–7 дней. Поэтому задача была максимально упрощена.

В качестве примера рассмотрим программу поиска НЛО (нужно учитывать, что в работе все цветные изображения переводятся в grayscale, иначе количество инвариантов слишком велико).

При условии, что:

– объект может иметь разный цвет ± 50 значений от исходного;

– объект может иметь разный размер, отличающийся в 3 и более раз;

– объект имеет разный угол наклона (угол колеблется в пределах 30°);

– объект имеет случайное месторасположение на изображении.

Этап 1. Создание обучающей выборки.

Первый и очень важный шаг – необходимо создать обучающую выборку. Здесь можно пойти двумя путями. Использовать для обучения заранее составленную базу изображений (например лиц) или же сгенерировать на основе одного эталонного объекта заданное количество случаев. Нам подходит последний вариант, тем более, что для генерации выборки на основе одного объекта в OpenCV есть модуль createsamples. В качестве фоновых изображений (т. е. изображений, на которых отсутствует искомый объект), используется пара сотен картинок космоса.

В зависимости от заданных параметров модуль берет эталонный объект, применяет к нему различные деформации (поворот, цвет, шум), далее выбирает фоновое изображение и располагает случайным образом на нем объект.

Этап 2. Обучение.

Теперь необходимо создать каскад классификаторов для имеющейся базы объектов. Для этого используем модуль haartraining. В него передается много параметров, самыми важными из которых являются количество классификаторов в каскаде, минимальный необходимый коэффициент эффективности классификатора (minimum hit rate), максимально допустимая частота ложных срабатываний (maximum false alarm).

Этап 3. Тестирование каскада.

Далее программа выдает каскад в виде xml файла, который может использоваться непосредственно для детекта объектов. Чтобы протестировать

его, мы снова генерируем 1000 объектов по принципу, описанному на первом этапе, создавая тем самым проверочную выборку.

Задание: создать программу для поиска своего собственного объекта на изображении.

Тема 16. Создания интерактивных тренажеров с использованием библиотек Irrlicht, ChronoEngine и OpenCV

Цель: формирование практических умений по созданию комплексных интерактивных тренажеров.

Теория и примеры выполнения задания

Для объединения всех умений и навыков, приобретенных в результате выполнения всех предыдущих практических заданий, построим простой тренажер гусеничного механизма.

Создадим упрощенную модель транспортного средства с треками, которые взаимодействуют с препятствиями, и элементами управления для пользователя.

Задание: создать симулятор для собственной модели.

ЛИТЕРАТУРА

1. Обработка изображений : ЭУМК [Электронный ресурс]. – 2011. – Режим доступа: http://bsuir.by.

2. Графический интерфейс интеллектуальных систем : ЭУМК [Электронный ресурс]. – 2010. – Режим доступа: http://bsuir.by.

3. Алгоритмы компьютерной гафики : ЭУМК [Электронный ресурс]. – Минск: БГУИР. – 2008. – Режим доступа: http://bsuir.by.

4. Блинова, Т. А. Компьютерная графика : учеб. пособие / Т. А. Блинова, В. Н. Порев; под ред. В. Н. Порева. – Киев : Юниор, 2006. – 520 с.

5. Зенкин, А. А. Когнитивная компьютерная графика. – М.: Наука. – 1991. – 187 с.

6. Кравченя, Э. М. Компьютерная графика : учеб. пособие / Э. М. Кравченя, Т. И. Абрагимович. – Минск : Новое знание, 2006. – 248 с.

7. Петров, М. Н. Компьютерная графика : учеб. пособие для вузов / М. Н. Петров, В. П. Молочков. – СПб. : Питер, 2002. – 735 с.

8. Лосик, Г. В. Перцептивные действия: кибернетический аспект. – Минск: ОИПИ, 2007. – 144 с.

9. Виртуальная реальность. Исторический обзор, киберпространство. [Электронный ресурс]. – 2010. – Режим доступа: http://virtual-real.info.

Св. план 2014, поз. 18

Учебное издание

Быков Антон Алексеевич
Мельникова Елена Александровна
Яшин Константин Дмитриевич

КОГНИТИВНЫЕ ТЕХНОЛОГИИ

ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *М. В. Гуртатовская*

Подписано в печать 19.06.2015. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,23. Уч.-изд. л. 5,4. Тираж 100 экз. Заказ 277.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6