Система интеграции алгоритмов обработки изображений, написанных на C++, с помощью интерпретируемых языков средствами .NET

Кравцов А.А., Юрин Д.В.

МГУ имени Ломоносова. Факультет ВМиК
Москва, 119899, Воробьевы горы, МГУ, 2-й учебный корпус
andrey.a.kravtsov@gmail.com, yurin_d@inbox.ru

Аннотация

Предложен подход, позволяющий строить сложные алгоритмы обработки изображений из составных блоков-частей, написанных на высокоэффективном C++ [1,2], используя средства платформы Microsoft .NET [3], включая встроенный интерпретатор языка JScript. Описан способ представления отдельных алгоритмов в виде динамически подгружаемых модулей .NET. Преимуществом системы является возможность конструирования алгоритма на высоком уровне и без перекомпиляции. Предложен формат описания объекта изображения, удобный для обмена данными между классами, выполняющимися в различных динамических библиотеках, и позволяющий избегать копирования изображений из одного контейнера в другой похожий. Приведен пример использования системы для решения задачи нахождения прямых линий на изображении.

1. ВВЕДЕНИЕ

Современные алгоритмы обработки изображений весьма сложны и включают в себя одновременно большое количество разнообразных алгоритмов более низкого уровня. Обычно в публикациях удается сформулировать алгоритм в виде псевдокода разумного размера [4-7]. Это говорит о том, что алгоритм может быть представлен как некоторый супералгоритм, строительными блоками которого являются алгоритмы более низкого уровня, такие, как преобразование Фурье или Хартли, интерполяция, фильтрация, поиск максимумов, дифференцирование (свертка) и т.д. Обычным подходом к программной реализации такого супералгоритма является реализация всех его строительных блоков на C++ и интеграция их в единую программу на C++, причем зачастую вместе с графическим интерфейсом [8]. Если выбор языка C++ для ряда низкоуровневых алгоритмов является единственно оптимальным, то этого нельзя сказать о построении супералгоритма, так как основное время программа проводит внутри низкоуровневых блоков, а реализация супералгоритма сводится к последовательному или циклическому вызову не более десятка низкоуровневых процедур. Сведение всех блоков в единый проект на C++ требует поддержки единых форматов данных и написания всех частей кода в единой манере. Кроме того, попытка использовать доступные готовые компоненты (алгоритмы),

выложенные авторами для всеобщего доступа, часто наталкивается на препятствия в виде различных способов представления изображений, разных правил кодирования или конфликта имен. Извлечение же из предоставленного кода нужных частей тоже часто затруднительно, так как в едином проекте вместе смешаны низкоуровневые алгоритмы, супералгоритмы, графические интерфейсы и форматы данных. В этой связи представляется целесообразным структурирование программ таким образом, чтобы использовать (максимально повторно) эффективные реализации низкоуровневых алгоритмов, работающих с пикселями непосредственно, супералгоритмы реализовывать через вызовы этих эффективных процедур с помощью интерпретируемого языка, а графический интерфейс реализовывать отдельно на оптимальном для данной операционной системы языке. В настоящее время на платформе Windows таким естественным средством является языки платформы .NET[3]. Интеграция с модулями на C++ естественным образом может быть выполнена через язык Managed C++, который позволяет создавать видимый для всех языков интерфейс в стиле .NET, и в свою очередь внутри такого модуля доступно все богатство выразительных средств C++, начиная от указателей и заканчивая шаблонами и полиморфизмом. Более того, в .NET уже встроен интерпретатор языков Jscript и VBasicScript (протранства имен Microsoft.Vsa, Microsoft.JScript, Microsoft.VBScript), расширенных до практически полной поддержки всех средств .NET.

Целью настоящей работы являлась разработка подхода, позволяющего использовать высокоэффективные коды на C++ для низкоуровневых (попиксельных) процедур, объединять их в единый супералгоритм с помощью интерпретируемых языков, не требующих компиляции и сделать эту часть доступной для правок пользователя и разумеется более прозрачной, избежать копирования изображений из одного контейнера в другой исключительно для обеспечения совместимости между модулями на C++, написанными разными людьми.

2. АРХИТЕКТУРА СИСТЕМЫ

Общая схема системы представлена на Рис. 1.

«Основное приложение» - приложение, написанное на одном из языков платформы .NET. Это может быть как уже существующее приложение, так и только разрабатываемое.

«Интерпретатор», или скриптовая машина, - класс C#, реализующий интерпретатор скриптов, написанных на интерпретируемом языке JScript.NET (или VBScript.NET). Входными данными скриптовой машины является непосредственно сам текст скрипта и ссылки на динамически подгружаемые библиотеки (если такие имеются). От основного приложения в скриптовую машину также могут передаваться как переменные фундаментальных типов, так и объекты .NET. Плюсом в данной ситуации является возможность передачи переменных любого из типов платформы .NET.

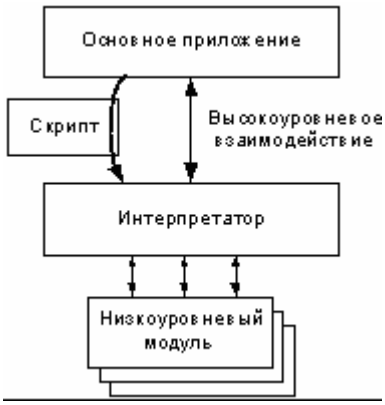

Рис.1. Архитектура системы

В тексте скрипта содержатся инструкции на языке JScript.NET: создание объектов, вызов функций из библиотек. Отметим некоторые преимущества использования скрипта: во-первых, все низкоуровневые манипуляции скрываются от разработчика основного приложения, для которого система становится черным ящиком, принимающим данные и выдающим результат, причем детали реализации не важны. Во-вторых, JScript.NET – простой, но в то же время богатый язык программирования, в котором доступны все возможности, предоставляемые платформой Microsoft .NET. В-третьих, при изменении текста скрипта не требуется перекомпиляции никакой из составных частей системы. Из последнего следует, что данная система предоставляет возможность программировать пользователю, который теперь свободен в выборе деталей реализации алгоритма. Единственное ограничение, которое существует у пользователя – это интерфейс взаимодействия с основным приложением.

«Низкоуровневый модуль» - динамически подгружаемая библиотека (dll), которая содержит класс на языке Managed C++, являющийся оболочкой для C/C++ кода. Использование управляемого класса гарантирует унификацию модуля, а реализация алгоритма на native-языке обеспечивает высокую скорость работы.

3. ОБЪЕКТ ИЗОБРАЖЕНИЯ

Для представления изображения в памяти был выбран формат, совместимый с используемым комплексом ENVI [9], который, по-видимому, отличается на сегодняшний день наибольшей общностью в смысле поддерживаемых типов данных и простотой. Отметим, что по организации данных этот формат включает в себя как частный случай такие контейнеры для изображений как DIB и BMP. Так как для алгоритмов обработки неважно происхождение изображения, а существенен только способ доступа к данным,

был выбран подход, позволяющий эффективно работать с изображениями, созданными внешними средствами, в частности в случае C# удается избежать копирования данных в другой контейнер.

Для описания внутренней организации данных объекта изображения был разработан native-класс ImageDescriptor, содержащий следующие открытые поля

Width – ширина изображения в точках;

Height – высота изображения в точках;

Bands – количество цветовых компонент (например, 1 – для полутонового изображения, 3 – для цветного RGB);

DataType – тип данных точки. Поддерживаемые типы: Byte (DataType = 1), unsigned short (16 bit, DataType = 12) и float32 (DataType = 4).

Interleaving – способ чередования цветовых компонент. Поддерживаемые значения: BSQ (Band sequentional) – сначала расположены все данные первой цветовой компоненты изображения, далее второй компоненты и т.д.; BIL (band interleaving by line) – чередование цветовых компонент построчно; BIP (band interleaving by pixel) – цветовые компоненты чередуются для каждого пиксела.

NLS (next line shift) – величина, которую необходимо прибавать к указателю, чтобы перейти в ту же позицию на изображении, но строчкой ниже. Может отличаться от Width (например, если идет работа с фрагментом изображения без копирования его в отдельный массив).

NBS (next band shift) – величина, которую необходимо прибавить к указателю, чтобы перейти в ту же позицию на изображении, но в следующей цветовой компоненте.

DataPtr – указатель на данные.

Экземпляры класса ImageDescriptor не содержат реальных данных, они лишь хранят полное описание изображения. Необходимость в разработке этого класса заключается в следующем: предполагается, что алгоритмы из различных динамически подргужаемых библиотек будут обмениваться между собой данными, в частности изображениями. Передавать непосредственно изображения, занимающие мегабайты памяти, неэффективно. Передавать указатель на класс ImageDescriptor между независимо собранными модулями рискованно из-за проблем выравнивания и версий компилятора, поэтому предусмотрен метод

int GetInfoRow(), возвращающий целое число (handle), фактически являющееся указателем на массив из восьми целых чисел – полей класса ImageDescriptor, и

конструктор *ImageDescriptor(int inforow)*, наполняющий по указателю поля класса – дескриптора.

За счет того, что тип данных int поддерживается как в языках C/C++, так и в платформе .NET и по размеру может вмещать указатель, такой способ описания изображений позволяет обмениваться данными в этих средах, а содержащийся в поле дескриптора C++ указатель на данные изображения является обычным C++ указателем, и в рамках одного приложения доступ до данных по нему является нормальной практикой C++.

4. ПРИМЕР

Для демонстрации изложенного выше подхода был выбран алгоритм отыскания прямых линий на изображении, описанный в [4]. Блок-схема алгоритма представлена на рис. 2.

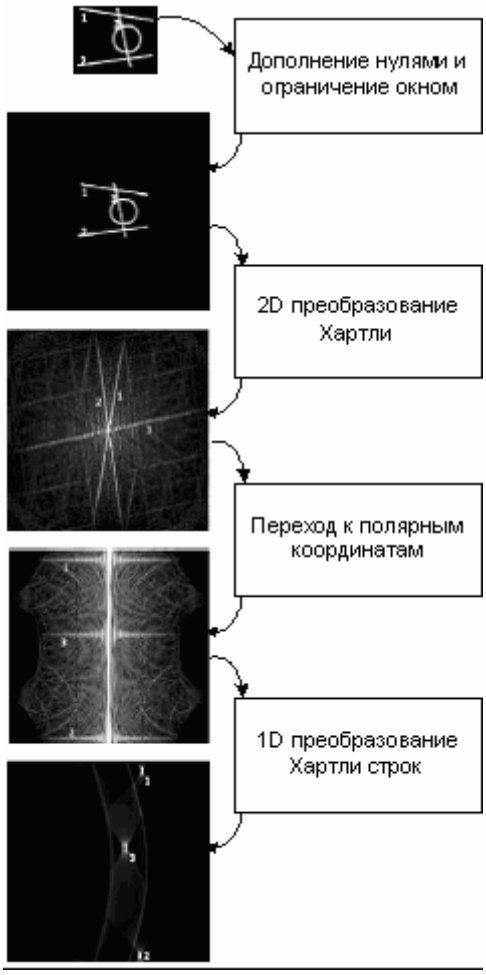

Рис.2. Блок-схема алгоритма нахождения прямых линий на изображении

4.1 Код в основном приложении

В следующем листинге представлена часть кода на языке C#, отвечающая за создание скриптовой машины, чтение текста скрипта из файла, интерпретацию (исполнение) скрипта:

C#
```
string appPath = Application.StartupPath;

m_ScriptEngine = ScriptEngine.GetInstance(new string[] {
    appPath + @"\DotNetImageWrapper.dll",
    appPath + @"\FilterPreprocessing.dll",
    appPath + @"\Hartley.dll",
    appPath + @"\Hartley1DWrapper.dll",
    appPath + @"\ToPolarWrapper.dll"});

string script = ScriptEngine.ReadScriptFromFile(Application.StartupPath+@"\LineDetection.js");
m_ScriptEngine.AddVariableAndInitialize("_filterInput", new Bitmap("demo.bmp"));

m_ScriptEngine.Eval(script);
m_ScriptEngine.Eval("JLastLineObject.Run();");
```

4.2 Интерпретатор

Ключевую роль в данной работе играет интерпретатор скриптов, написанных на языке JScript.NET. При разработке программы были использованы сборки Microsoft.JScript.dll и Microsoft.Vsa.dll. К сожалению, в стандартной документации [10] возможности классов из пространств имен Microsoft.JScript и Microsoft.Vsa не описаны, в связи с чем о существующих функциях приходится догадываться лишь по названиям классов, методов и свойств.

В связи с важностью класса интерпретатора и проблематичностью его реализации целесообразно привести полный код класса ScriptEngine:

C#
```
using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using System.Windows.Forms;
using Microsoft.JScript;
using Microsoft.JScript.Vsa;
using Microsoft.Vsa;

public class ScriptEngine :
System.ComponentModel.Component
{
GlobalScope m_GlobalScope = null;

public GlobalScope GlobalScope
{
    get { return m_GlobalScope; }
}

// Конструктор. Принимает массив полных имен
// сборок, которые необходимо загрузить
public ScriptEngine(string[] assemblyReferences)
{
  GlobalScope gs =
     VsaEngine.CreateEngineAndGetGlobalScope(
        false, assemblyReferences);

  VsaEngine engine = gs.engine;

  GlobalScope newGS =new GlobalScope(gs,engine);

  engine.PushScriptObject(newGS);
  m_GlobalScope = newGS;
}

// Интерпретация скрипта
public object Eval(string scriptText)
{
  // возвращаемый скриптом объект
  object result = null;
  try
  {
    result = Eval.JScriptEvaluate(scriptText,
         m_GlobalScope.engine);
  }
  catch (JScriptException ex)
  {
    result = ex;
    MessageBox.Show(ex.Message, string.Format(
      "Error in script in position ({0}, {1})",
      ex.Line.ToString(),ex.Column.ToString()),
      MessageBoxButtons.OK,
      MessageBoxIcon.Error);
  }
  catch (Exception ex)
  {
     MessageBox.Show(ex.Message, "Error",
     MessageBoxButtons.OK,
     MessageBoxIcon.Error);
   }
   return result;
}

// Чтение текста скрипта из файла
public static string ReadScriptFromFile(
   string fileName)
{
    System.IO.StreamReader sr = new
       System.IO.StreamReader(fileName);
    string sScript = "";
    try
    {
        sScript = sr.ReadToEnd();
    }
    catch (Exception ex)
    {
        //TODO
    }
    finally
    {
      sr.Close();
    }
```

```
  return sScript;
}
// Функция добавления всех пространств имен из
// указанной сборки
static void CollectNamespaces(Assembly assembly,
                              Hashtable hTable)
{
    Type[] types = assembly.GetTypes();
    if (types == null || types.Length == 0)
        return;

    foreach (Type type in types)
    {
        if (!type.IsSubclassOf(typeof(Object)))
            continue;

        if (type.Namespace != null)
            hTable[type.Namespace] = type;
    }
}

// Функция добавления в таблицу hTable всех
// сборок необходимых для работы сборки assembly
static void CollectReferences(Assembly assembly,
Hashtable hTable)
{
    hTable[assembly.FullName] = assembly;

    AssemblyName[] aNames =
       assembly.GetReferencedAssemblies();

    if (aNames == null || aNames.Length == 0)
        return;

    foreach (AssemblyName aName in aNames)
    {
        string fullName = aName.FullName;

        if (hTable.ContainsKey(fullName))
            continue;

        try
        {
            Assembly assemb =
               Assembly.Load(aName);
            hTable.Add(fullName, assemb);
        }
        catch { }
    }
}

// Загрузка пространств имен
void Import(params string[] list)
{
    foreach (string listItem in list)
    {
        Import.JScriptImport(listItem,
           GlobalScope.engine);
    }
}
// Создание переменной name в области видимости
// скрипта
public void AddVariable(string name)
{
    m_GlobalScope.AddField(name);
}

// Присвоение переменной name значения value
public object SetVariable(string name,
                          object value)
{
   return m_GlobalScope.InvokeMember(name,
     BindingFlags.SetField, null, m_GlobalScope,
     new object[] { value }, null, null, null);
}
// Объявление и инициализация переменной
public void AddVariableAndInitialize(string
variableName, object variableValue)
{
    AddVariable(variableName);
    SetVariable(variableName, variableValue);
}
// Чтение значения переменной varName
public object GetVariable(string varName)
{
   return m_GlobalScope.InvokeMember(varName,
     BindingFlags.GetField, null, m_GlobalScope,
     null, null, null, null);
}

//Метод, возвращающий экземпляр класса
//ScriptEngine. Принимает массив путей к сборкам
public static ScriptEngine GetInstance(
   string[] dlls)
{
   List<Assembly> aAssemblies =
      new List<Assembly>();

   // добавление в коллекцию всех загруженных
   // сборок
   aAssemblies.AddRange(
      AppDomain.CurrentDomain.GetAssemblies());

   foreach (string dll in dlls)
   {
      try
      {
      // Загрузка всех dll
      Assembly assembly =Assembly.LoadFrom(dll);

          if (!aAssemblies.Contains(assembly))
              aAssemblies.Add(assembly);
      }
      catch { }
   }

   Hashtable tableAssemblies = new Hashtable();

   // Создание коллекции необходимых сборок
   foreach (Assembly assembly in aAssemblies)
   {
      ScriptEngine.CollectReferences(assembly,
         tableAssemblies);
   }

   Hashtable tableNamespaces = new Hashtable();

   // Создание коллекции пространств имен
   foreach (DictionaryEntry entry in
       tableAssemblies)
   {
      ScriptEngine.CollectNamespaces(
        (Assembly)entry.Value, tableNamespaces);
   }

   string[] aStringDll = (string[])(
   new ArrayList(tableAssemblies.Keys)).ToArray(
       typeof(string));

   string[] aStringNS = (string[])(
   new ArrayList(tableNamespaces.Keys)).ToArray(
       typeof(string));

   ScriptEngine engine = new
      ScriptEngine(aStringDll);
   engine.Import(aStringNS);
   return engine;
}
}
```

4.3 Низкоуровневые модули

Как уже было сказано выше, все алгоритмы инкапсулированы в динамически подгружаемые библиотеки .NET. В качестве примера приведем код managed-оболочки для двумерного преобразования Хартли:

```
Managed C++
// include section
#include "integral_transforms.h"

using namespace System;
using namespace System::Drawing;
using namespace ScriptImaging;

public ref class Hartley2DWrapper
{
public:
   Hartley2DWrapper(){}
   ~Hartley2DWrapper(){}
   !Hartley2DWrapper() {}
public:
   void DoJob(int imginfo)
   {
        ImageDescriptor img_id(imginfo);

        size_t n[] = {img_id.Width,img_id.Height};

        float* fPtr = (float*)(img_id.DataPtr);

        SwapQuadrants<2, false>(fPtr, n);
        fht2D(fPtr, n[0], n[1]);
        SwapQuadrants<2, false>(fPtr, n);
   }
};
```

Из этого примера видно, насколько просто создать managed-оболочку на функции C++, коими тут являются SwapQuadrants и fht2D.

4.4 Скрипт

В следующем листинге приведен текст скрипта, в котором над изображением производятся преобразования, описанные в блок-схеме на Рис.2:

```
JScript.NET
public class JLineDetector
{
    public function Run(): void
    {
        var bmpW:int = _filterInput.Width;
        var bmpH:int = _filterInput.Height;

        var maxWH:int = Math.max(bmpW, bmpH);
        var twoPow:int = 1;

        while (twoPow < maxWH)
            twoPow *= 2;

        var img1:ScriptImage = new ScriptImage(twoPow, twoPow, 1, 3);
        var img2:ScriptImage = new ScriptImage(twoPow, twoPow, 1, 3);
        var filter1:ZeroPadding = new ZeroPadding();
        filter1.DoJob(_filterInput, img1.InfoRow);
        var filter2:Hartley2DWrapper = new Hartley2DWrapper();
        filter2.DoJob(img1.InfoRow);
        var filter3:ToPolarWrapper = new ToPolarWrapper();
        filter3.DoJob(img1.InfoRow, img2.InfoRow);
        var filter4:Hartley1DWrapper = new Hartley1DWrapper();
        filter4.DoJob(img2.InfoRow);
        // Save images
        img1.SaveAsBitmap(false, "out1.png");
        img2.SaveAsBitmap(false, "out2.png");
    }
};

var JLastLineObject:JLineDetector = new JLineDetector();
```

4.5 Передача изображений между языками

Отметим важную особенность представления изображения и способа его передачи между языками: объект ScriptImage создается внутри скрипта (JScript.NET), передается в функцию из библиотеки как целое число (по сути handle). Внутри функции по этому числу создается дескриптор изображения ImageDescriptor, с помощью которого можно получить доступ к таким свойствам изображения, как ширина, высота, указатель на данные и др. Затем в случае необходимости можно создать класс-оболочку, манипулирующий внешними данными.

Литература.

1. Александреску А. Современное проектирование на C++. Серия C++ In-Depth, т.3.:Пер. с англ. -М.:Издательский дом "Вильямс", 2002. -336 с.
2. Страуструп Б. Язык программирования C++. -Минск: BHV, 2008, 1104 с.
3. Троелсен Э., C# и платформа .NET, -СПб.: Питер, 2006, 752 с.
4. D.B. Volegov, V.V. Gusev, D.V. Yurin *Straight Line Detection on Images via Hartley Transform. Fast Hough Transform.* //16-th International Conference on Computer Graphics and Applications GraphiCon'2006. Novosibirsk Akademgorodok, Russia, July 1 - 5, 2006, -P. 182-191. http://www.graphicon.ru/2006/proceedings/papers/fr11_35_Volegov_Gusev_Yurin.pdf.
5. B.S. Reddy, B.N. Chatterji, "An FFT-based technique for translation, rotation, and scale-invariant image registration", IEEE PAMI, Vol 5(8), pp. 1266-1271, August, 1996.
6. James Davis. Mosaics of scenes with moving objects. In Proc. Computer Vision and Pattern Recognition Conf., pages 354--360, 1998. http://citeseer.ist.psu.edu/davis98mosaics.html.
7. Siavash Zokai, George Wolberg. Image Registration Using Log-Polar Mappings for Recovery of Large-Scale Similarity and Projective Transformations //IEEE Transactions on Image Processing, Vol. 14, No. 10, October 2005. http://www-cs.engr.ccny.cuny.edu/~wolberg/pub/tip05.pdf
8. B. Georgescu, P. Meer. Point Matching under Large Image Deformations and Illumination Changes //IEEE Transactions On Pattern Analysis and Machine Intelligence, June 2004, -V. 26, -No. 6, -P. 674-688. Code available at http://www.caip.rutgers.edu/riul/research/code.html
9. Сайт компании ITT Visual Information Solutions, (before May 15, 2006 – RSI - Research Systems, Inc.), разработчик пакета ENVI, http://www.ittvis.com/index.asp.
10. Microsoft Developer Network (MSDN)