ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 200 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Денисов Е.Ю., Волобой А.Г., Калугина И.А.

# Автоматизация тестирования интерактивной системы моделирования освещенности

***Рекомендуемая форма библиографической ссылки:*** Денисов Е.Ю., Волобой А.Г., Калугина И.А. Автоматизация тестирования интерактивной системы моделирования освещенности // Препринты ИПМ им. М.В.Келдыша. 2018. № 200. 19 с. doi:10.20948/prepr-2018-200
URL: http://library.keldysh.ru/preprint.asp?id=2018-200

**Ордена Ленина**
**ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ**
**имени М.В.Келдыша**
**Российской академии наук**

**Е.Ю. Денисов, А.Г. Волобой, И.А. Калугина**

# Автоматизация тестирования интерактивной системы моделирования освещенности

**Москва — 2018**

*Денисов Е.Ю., Волобой А.Г., Калугина И.А.*

**Автоматизация тестирования интерактивной системы моделирования освещенности**

В статье описана технология автоматического регрессионного тестирования, применяемая при разработке и сопровождении в пределах жизненного цикла современных программных комплексов компьютерной графики и оптического моделирования. Рассмотрены основные типы ошибок и способы реализации тестов с помощью командной строки, языка сценария Python и пакета AutoIt, позволяющего создавать тесты для интерактивного графического интерфейса программы. Применение описанной технологии на практике позволяет снизить время тестирования выпускаемой версии программного продукта с нескольких недель до двух-трех часов.

***Ключевые слова:*** компьютерная графика, оптическое моделирование, автоматическое тестирование, регрессионное тестирование

*Evgeny Yurievich Denisov, Alexey Gennadievich Voloboy, Irina Aleksandrovna Kalugina*

**Automatic testing of interactive lighting simulation software package**

The article is devoted to automatic regression testing technology used in development and maintenance within the life cycle of modern computer graphics and optical simulation software systems. The main types of errors and ways to implement tests using the command line, the Python script language, and the AutoIt package, which allow you to create tests for an interactive graphical user interface, are considered. In practice the described technology allows to reduce the time for testing the released version of a software product from several weeks to couple of hours.

***Keywords:*** computer graphics, lighting simulation, automatic testing, regression testing

# Оглавление

# Введение

Процесс создания современных программных систем требует автоматизации их разработки и сопровождения на всех этапах жизненного цикла программного продукта. Одним из важнейших этапов является тестирование. Сложность современных программных комплексов оптического моделирования, созданных нашим коллективом, достигла уровня, при котором классическое тестирование [1] становится неэффективным, а часто и невозможным по ряду причин. Можно выделить следующие особенности тестирования таких комплексов:

- разнородные программные комплексы используют общие компоненты;
- частые выпуски новых версий;
- ограниченные ресурсы и время.

В этих условиях большую роль в повышении эффективности разработки играет автоматизация тестирования программ. В основном используются два типа тестирования: тестирование на основе аналитических расчётов, обеспечивающее сравнение результатов работы системы с ожидаемыми, и так называемое регрессионное тестирование, обеспечивающее совпадение поведения новой реализации системы и её предыдущей реализации.

В частности, было неоднократно замечено, что ошибки в программе, однажды найденные и исправленные, могут иногда возникнуть вновь в будущих версиях программных продуктов. Заметим, что с точки зрения конечного пользователя такие инциденты подрывают доверие как к конкретному программному продукту, так и к квалификации коллектива разработчиков в целом, а значит, и к качеству других продуктов, созданных этим коллективом.

Необходимо отметить, что причинами повторного появления таких вроде бы уже однажды исправленных ошибок могут быть как банальные ошибки, допущенные в процессе интеграции исходных кодов, так и совершенно новые независимые ошибки, которые по случайному стечению обстоятельств приводят к такому же внешнему проявлению, как и ранее исправленные ошибки. Однако для конечного пользователя, которому неизвестны истинные причины повторного появления ошибки, эти причины не имеют особого значения – факт повторного проявления уже однажды найденной и исправленной ошибки неизменно отрицательно влияет как на репутацию программного продукта, так и на репутацию коллектива его разработчиков.

Не рассматривая здесь меры по предотвращению подобных ситуаций, опишем подход, позволяющий выявить такие повторные появления уже однажды исправленных ошибок. Подход этот состоит в проверке каждой вновь выпускаемой версии программного продукта на отсутствие всех ранее зарегистрированных и уже однажды исправленных ошибок. Такая проверка позволяет избежать повторного проявления ошибок вне зависимости от их

причины. Такое тестирование является регрессионным, поскольку позволяет удостовериться в совпадении поведения новой версии программы и её предыдущей версии.

## Типы ошибок ПО и способы их выявления

Исходя из нашего опыта разработки, можно выделить следующие типы встречающихся ошибок:

- ошибочные результаты расчётов, представляемых в численном виде (неверное вычисление);
- ошибочные результаты визуализации (неверное отображение результатов в графическом виде);
- потеря скорости расчётов (программа затрачивает на расчёт значительно большее время, чем раньше);
- потеря эффективности использования памяти (программа необоснованно затрачивает для расчёта большее количество оперативной памяти, чем раньше);
- «зависание» программы в процессе работы, вызываемое определённым набором входных данных либо определёнными действиями пользователя;
- аварийное завершение работы программы, вызываемое определённым набором входных данных либо определёнными действиями пользователя.

Для каждой найденной в программном продукте и исправленной ошибки создаётся отдельный тест, позволяющий в автоматическом режиме проверить отсутствие указанной ошибки в тестируемом продукте. Таким образом, система тестирования должна быть способна делать вывод о наличии или отсутствии ошибки на основе:

- анализа или сравнения численных результатов расчёта;
- анализа или сравнения графических результатов расчёта;
- анализа скорости работы программы;
- анализа объёма памяти, используемой программой в процессе расчёта;
- слежения за состоянием расчётных процессов для определения их «зависания» или аварийного завершения.

## Особенности автоматизации тестирования

Программные продукты, разрабатываемые нашим коллективом, содержат в своём составе модули, предоставляющие необходимую функциональность в нескольких формах. Как правило, это:

- модуль пакетной обработки данных (или модуль командной строки);
- модуль API, предоставляемый как пакет для языка программирования Python;
- интерактивная программа с графическим интерфейсом пользователя.

Это разнообразие эффективно достигается путём использования общего вычислительного ядра системы с различными модулями интерфейса. Ошибки встречаются во всех модулях, поэтому система тестирования должна поддерживать все три указанных функциональности. Для ошибок, воспроизводимых только в какой-либо одной из вышеуказанных форм, должно применяться тестирование именно этой формы. Однако в случаях, когда ошибка может быть воспроизведена в любой форме, для её тестирования выбирается та форма, в которой тест может быть написан наиболее просто и эффективно. Например, ошибка в алгоритме расчёта освещённости, допущенная внутри общего вычислительного ядра, будет, очевидно, проявляться независимо от используемого интерфейса, и тест на отсутствие этой ошибки может быть написан с использованием самой простой формы – модуля командной строки.

Результатом каждого автоматического теста является решение тестирующей системы о том, пройдён ли тест успешно или нет. По окончании работы выдаётся таблица с результатами всех тестов, что позволяет сделать вывод либо об успешном прохождении этапа автоматического тестирования, либо о необходимости исправления найденных ошибок.

Для облегчения анализа результатов в случае неудачного теста при возможности также автоматически создаются изображения – ожидаемое и полученное, с количественным и качественным описанием разницы между ними.

Рассмотрим более подробно методы и приёмы тестирования, применяемые для каждой из описанных форм.

## Тестирование модуля командной строки

Так как создание теста для модуля командной строки является наиболее простой задачей с технической точки зрения, этот тип тестов является наиболее предпочтительным всегда, когда ошибка может быть воспроизведена в этой форме, даже если изначально эта ошибка была обнаружена, например, при использовании программы с графическим интерфейсом. Тестирование с использованием модуля командной строки состоит, как правило, в подготовке исходных данных, выполнении необходимого расчёта и анализе результатов. Численные результаты могут быть проанализированы сравнением их с результатами, заранее рассчитанными либо аналитическим методом, либо предыдущей, заведомо верной версией программы. Графические результаты анализируются путём сравнения полученных изображений с изображениями, полученными предыдущей версией программы.

Необходимо учитывать, что расчёты ведутся с использованием стохастических методов и полученные изображения будут различаться (как правило, незаметно для глаза). Поэтому сравнение изображений проводится с определённым уровнем точности, величина которого зависит от тестируемой подсистемы; для сравнения изображений создана программа, выдающая в

качестве результата разницу между пикселями сравниваемых изображений. Разница выдаётся как в числовом формате, характеризующем разницу между пикселями (как абсолютную, так и относительную), так и в графическом, в виде изображения, где большая яркость пикселей соответствует большей разности в этом месте между сравниваемыми изображениями.

Кроме контроля результатов, модуль командной строки используется для контроля скорости работы программы (путём вычисления времени, затраченного модулем на расчёт), а также для проверки отсутствия «зависания» и аварийного завершения программы при определённых исходных данных (путём контроля наличия и своевременного завершения соответствующих процессов).

## Тестирование с использованием Python API

Использование модуля с интерфейсом Python API является следующим по предпочтительности и используется в тех случаях, когда модуль командной строки не может быть использован, как правило, из-за недостаточной функциональности или отсутствия некоторых тонких настроек. Этот модуль, за счёт расширенных возможностей, предлагаемых программным API комплекса [2] и самим языком программирования Python, позволяет проверить все те же типы ошибок, что и модуль командной строки, а также дополнительно проверить количество памяти, занимаемое программой в процессе работы, отсутствие «зависания» и аварийного завершения программы в результате определённой последовательности команд.

Для проверки количества памяти, занимаемой программой в процессе расчёта, проводится определённое число одинаковых операций, захватывающих и потом освобождающих память, и сравнивается объём памяти, занимаемый программой после каждой операции. Таким образом, постоянное увеличение количества занятой памяти после каждой операции однозначно указывает на ошибку типа «утечка памяти».

## Тестирование пользовательского интерфейса

Этот тип тестирования является наиболее трудозатратным в смысле создания тестов и применяется, только если ни модуль командной строки, ни модуль Python API не могут быть использованы. На основании накопленного опыта тестирования пользовательского интерфейса наших программных комплексов оптического моделирования нами были выдвинуты следующие требования к системе автоматического тестирования:

- способность распознавать элементы пользовательского интерфейса (окна, диалоги, кнопки, поля ввода, чекбоксы, слайдеры и т.д.) и производить над ними действия (ввод символов с клавиатуры, одиночное и двойное нажатие левой, правой и средней кнопками мыши, операции с колёсиком мыши, передвижение окон и диалогов, ввод и считывание значений);

- поддержка продвинутого языка сценариев, позволяющего производить математические операции над числовыми величинами и операции над текстовыми строками;
- возможность внесения изменений в имеющийся сценарий без полной его перезаписи;
- поддержка работы с текстовыми файлами (чтение, запись);
- отсутствие необходимости изменения кодов тестируемой программы (например, включение вспомогательных кодов для взаимодействия с внешним тестирующим пакетом);
- способность оперировать относительными координатами (привязанными к окну тестируемой программы) вместо абсолютных (привязанных к экрану);
- способность оперировать выполняемыми процессами (по крайней мере, получение информации о них);
- наличие операций с таймерами (вычисление разницы во времени между событиями, внесение определённой задержки между действиями и т.д.).

Авторами были протестированы более семидесяти программных пакетов, позволяющих в том или ином виде записать и воспроизвести действия пользователя в интерактивной программе. Среди них: Test Automation FX [3], TestComplete [4], Squish [5], Sikuli [6], Atoma [7] и другие.

К сожалению, обнаруженные в большинстве пакетов недостатки не позволили полноценно использовать их для тестирования создаваемых программных комплексов. Среди этих недостатков:

- запись действий пользователя в абсолютных координатах экрана (не позволяет воспроизводить записанную сессию на другом компьютере с другим разрешением экрана);
- неспособность распознать UI элементы пакета Qt – основного средства разработки пользовательского интерфейса наших программных продуктов;
- отсутствие возможности внесения изменений в записанную сессию;
- отсутствие поддержки новейших версий Windows;
- ошибки в работе, иногда приводящие к сбою всей OS.

Рассмотрим в качестве примера недостатки двух из вышеупомянутых продуктов.

**Sikuli**

Пакет автоматизации Sikuli [6] является чисто графическим средством. Он оперирует с объектами пользовательского интерфейса, как с изображениями. Для того чтобы имитировать нажатие кнопки, необходимо создать (захватить) изображение этой кнопки и вставить его в скрипт. При этом качество распознавания изображения недостаточное. В случае наличия на экране двух объектов интерфейса с похожими надписями, например, кнопок с цифрами,

Sikuli может нажать «не ту» кнопку. В случае, если на экране видны две одинаковые кнопки (например, кнопки «Close» в двух разных одновременно открытых диалоговых окнах), то не определено, какая именно из кнопок будет нажата, и нет никакой возможности уточнения. Кроме того, отсутствует обратная связь, нет возможности определить реакцию на воздействие на объекты интерфейса. Например, определить, открылся ли необходимый диалог и какие в нём есть объекты интерфейса, не представляется возможным.

**Atoma**

Пакет автоматизации Atoma [7], к сожалению, не распознаёт объекты пользовательского интерфейса, созданные с применением пакета Qt – основного пакета, используемого в наших программных комплексах для создания пользовательского интерфейса. Это делает невозможным взаимодействие тестирующей системы с объектами пользовательского интерфейса тестируемой программы.

Тем не менее удалось подобрать один пакет, удовлетворяющий большинству условий. Это AutoIt [8] – бесплатный пакет для автоматизации действий с пользовательским интерфейсом в Windows.

**AutoIt**

AutoIt – это пакет автоматизации действий в пользовательском интерфейсе Windows, и не только. В его основе лежит язык сценариев, напоминающий BASIC. AutoIt использует комбинацию симуляции нажатий клавиш, движения мыши, операций с окнами и диалогами для автоматизации задач, не достижимой другими средствами (например, VBScript, SendKeys и т.д.). AutoIt – это небольшой по размеру самодостаточный пакет, работающий на всех версиях Windows и не требующий установки никаких дополнительных средств.

Изначально AutoIt был создан для автоматического конфигурирования тысяч компьютеров. Со временем он превратился в мощный язык сценариев, поддерживающий комплексные выражения, пользовательские функции, циклы и всё прочее, что могут ожидать специалисты от языка сценариев. Перечислим его основные особенности:

- простой для изучения синтаксис, похожий на язык программирования BASIC;
- имитация нажатий клавиш и движений мыши;
- манипуляция окнами и процессами;
- взаимодействие со всеми стандартными элементами пользовательского интерфейса Windows;
- возможность создания собственных элементов пользовательского интерфейса;
- поддержка технологии COM;
- поддержка регулярных выражений;

- прямой вызов функций из внешних и системных DLL;
- совместимость с Windows XP / 2003 / Vista / 2008 / Windows 7 / 2008 R2 / Windows 8 / 2012 R2 / Windows 10;
- поддержка Unicode строк;
- поддержка 64-битной архитектуры;
- сценарии могут быть скомпилированы в выполняемые модули;
- подробное руководство пользователя.

AutoIt занимает мало места и не требует внешних DLL файлов и записей в системном реестре, что делает его безопасным для выполнения на серверных версиях Windows. В комплекте поставляется редактор скриптов с поддержкой цветового выделения конструкций языка.

Много усилий было направлено на оптимизацию функций симуляции нажатий клавиш и движения мыши с целью сделать их максимально точными и надёжными во всех поддерживаемых версиях Windows. Все функции, относящиеся к контролю мыши и клавиатуры, имеют широкие возможности для настройки скорости и функциональности их действий. Функции взаимодействия с окнами и диалогами позволяют, в том числе, передвигать, прятать, показывать, изменять размер, активировать, закрывать окна и диалоги. Нужные окна могут быть определены (найдены) по их заголовку, содержимому, размеру, положению, классу и даже внутренним дескрипторам Win32 API.

Кроме операций с имеющимися окнами, AutoIt имеет средства для создания собственных окон и диалогов – поддерживаются все стандартные элементы пользовательского интерфейса Windows.

**Применённые решения**

Так как одни и те же элементы пользовательского интерфейса используются в различных программных комплексах, оказалось целесообразным создать библиотеку функций на языке сценариев AutoIt для упрощённого выполнения некоторых часто используемых команд, таких как «открыть файл», «сохранить результат расчёта» и т.д. Вот так, например, в этом языке сценариев выглядит функция для загрузки сцены – модели данных, состоящей из многих объектов различных типов, связанных определёнными иерархическими связями (геометрические объекты, свойства их поверхностей, источники света и т.д.):

```
; Function to open given scene
Func SceneOpen($scene_name)
  ; Bring window of our program to top
  WinActivate("Layouter")

  ; Send Ctrl+o
  Send("^o")
  ; Wait for "Open Scene" dialog to appear
  If WinWaitActive("Open Scene", "", $Timeout) = 0 Then
```

```
    Report("Dialog 'Open Scene' not found")
    Exit(1)
  EndIf

  ; Type scene filename
  Send($scene_name)
  ; Click "Open" button
  ControlClick("[ACTIVE]", "", "Open")
  ; Make sure scene was loaded,
  ; its name must be presented in window title
  CheckTitle($scene_name)
EndFunc
```

Кроме часто используемых простых команд, в библиотеку включены также сложные последовательности действий, выполняющие определённую часто используемую операцию, например, присваивание выбранной поверхности геометрического объекта сцены указанного свойства (цвет, прозрачность, текстуру и т.д.). Рассмотрим соответствующий пример – присваивание указанному объекту сцены текстуры с заданным разрешением по X и Y:

```
; Function to assign given texture with given size to given object
Func SetTexture($node, $tex_name, $size)
  ; Double click to open properties window
  ControlClick("Scene Hierarchy", "", $node, "left", 2)
  If WinWaitActive("Surface Properties", "", $Timeout) = 0 Then
    Report("Dialog 'Surface Properties' not found")
    Exit(1)
  EndIf

  ; Click on "Texture" tab
  ControlClick("Surface Props","","tab control","left",1,230,12)
  Sleep(1000)

  ; Click "Initial path"
  ControlClick("Surface Properties", "", "start_cmb")
  Send("{DOWN 5}{UP}{ENTER}")

  ; Load texture file
  ControlClick("Surface Properties", "", "Load texture")
  If WinWaitActive("Load texture file", "", $Timeout) = 0 Then
    Report("Dialog 'Load texture file' not found")
    Exit(1)
  EndIf
  Send($tex_name & "{ENTER}")

  ; Click "Image size" control
  ControlClick("Surface Props","","image size x","left",2)
  Send("100{TAB}");
```

```
  ; Pause for recalculation
  Sleep(2000)  Send("100{TAB}");
  ; Pause for recalculation
  Sleep(2000)

  ; Close dialogs
  ControlClick("Surface Properties", "", "OK")
  Sleep(1000)
EndFunc
```

Тестирование с использованием пакета AutoIt происходит следующим образом: для каждой функциональности тестируемой программы создан сценарий, который выполняется, воздействуя на элементы пользовательского интерфейса этой программы. Результат выполнения сценария сравнивается с ожидаемым, и при их совпадении тест считается успешным. Таким результатом, как правило, является:

- определённое состояние элементов интерфейса (необходимые диалоговые окна открыты, кнопки нажаты, текстовые поля имеют определённые значения) в ответ на интерактивные действия пользователя;
- правильное изображение в окне программы после определённого набора действий (например, изменение мощности источника света должно приводить к изменению освещённости объектов сцены);
- правильные численные значения проверяемых параметров (например, установка конкретного значения отражающей способности некоторого объекта должна приводить к известному, заранее посчитанному аналитически, количеству света, отражаемого этим объектом);
- правильная, т.е. заранее ожидаемая, реакция программы на определённые действия пользователя, как правильные, так и ошибочные (например, сообщение об ошибке в случае задания недопустимых параметров);
- правильный результат расчёта освещённости как для всей сцены, так и для определённых её участков (как правило, чувствительных к программным ошибкам) – определяется сравнением финального изображения с ожидаемым, и при разнице изображений свыше допустимой тест считается неуспешным;
- отсутствие ошибочной реакции программы (например, такой как аварийное завершение работы) в результате определённых действий.

К сожалению, пакет AutoIt имеет и некоторые недостатки, специфичные для наших программных комплексов. В нём не полностью реализована поддержка пакета GUI SDK «Qt», что приводит к некоторым неудобствам в работе. Например, отсутствует поддержка элементов меню (для вызова пунктов меню приходится использовать клавиатурные команды: Ctrl+o для открытия файла и т.д.), нет возможности захвата (для дальнейшей обработки или записи в файл) изображения с области экрана, нет возможности узнать состояние

некоторых элементов интерфейса (например, текущий элемент, выбранный в выпадающем списке).

В то же время AutoIt обладает рядом достоинств, которые позволяют существенно расширить область его применения. Его язык сценариев включает в себя наборы функций для:

- получения и изменения значений переменных среды окружения;
- чтения и записи текстовых и бинарных файлов;
- воспроизведения мультимедиа файлов;
- передачи данных по локальной сети;
- поддержки COM;
- управления процессами.

Кроме того, имеется мощнейший инструментарий для создания собственных элементов интерфейса, например, с целью интерактивного отображения процесса тестирования и управления им.

Кроме собственно тестирования, пакет AutoIt также может использоваться для демонстрации возможностей приложений и в качестве интерактивного дополнения к инструкции по эксплуатации приложения.

## Примеры тестирования

Ниже приведены несколько типичных результатов тестирования.

Результат теста на совпадение результатов расчёта с заранее рассчитанными аналитическими значениями выглядит следующим образом:

```
Expected value = 6.978e+03
Obtained value = 6.978e+03
Relative error = 0.0006675%
---------------------------
Status: passed
```

Рассмотрим несколько примеров, в которых вывод о корректности работы системы делается на основе сравнения изображений, полученных текущей и предыдущей версиями программы.

На рис. 1 приведено эталонное изображение, полученное методом BRT (Backward Ray Tracing, или метод обратной трассировки лучей) в предыдущей версии программы, а на рис. 2 изображение получено тем же методом в текущей (проверяемой) версии.

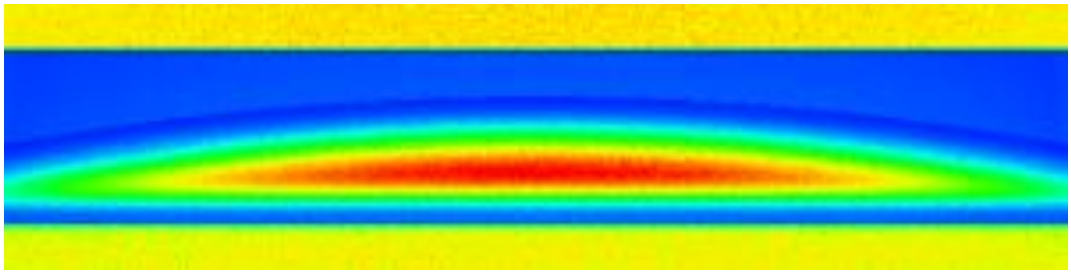

Рис. 1. BRT, эталонное изображение

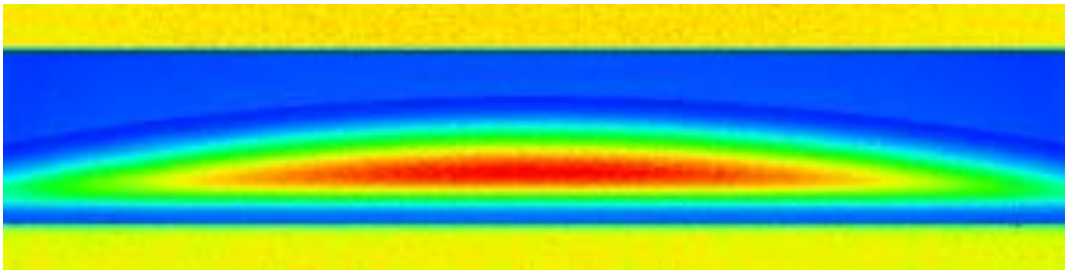

Рис. 2. BRT, проверяемое изображение

Разница между изображениями не должна превышать определённого значения:

```
RMS between images = 0.3356%
----------------------------
Status: passed
```

На рис. 3 приведено эталонное изображение, полученное методом PT (Path Tracing, или метод трассировки путей) в предыдущей версии программы, на рис. 4 изображение получено тем же методом в текущей (проверяемой) версии.

Рис. 3. PT, эталонное изображение

Рис. 4. PT, проверяемое изображение

Разница между изображениями не должна превышать определённого значения:

```
RMS between images = 0.1723%
----------------------------
Status: passed
```

Следующий пример – сравнение двух изображений, полученных с использованием различных технологий трассировки лучей. На рис. 5 показано изображение, полученное методом BMCRT (Backward Monte Carlo Ray Tracing, или обратная трассировка лучей методом Монте-Карло), на рис. 6 изображение получено методом PT (трассировка путей).

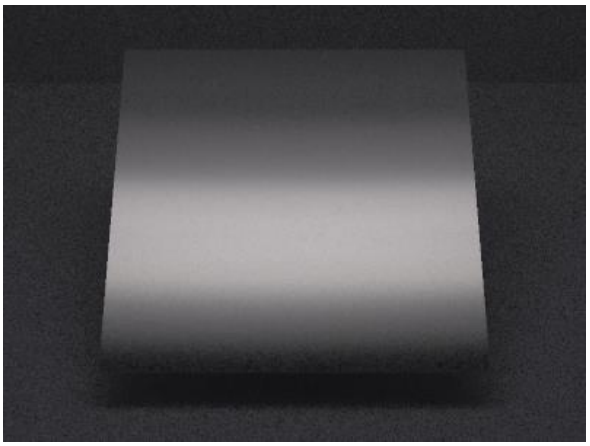

Рис. 5. Метод BMCRT

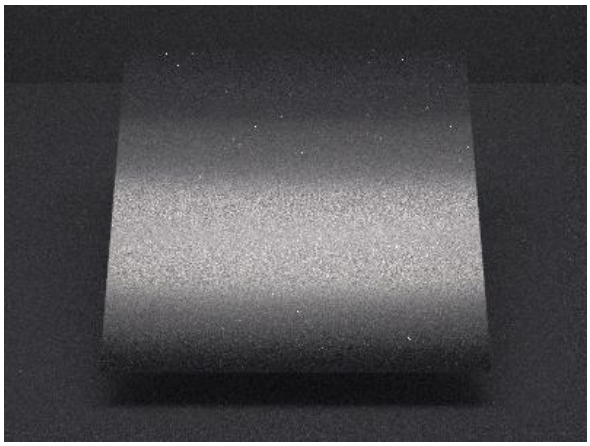

Рис. 6. Метод PT

Изображения различаются – это обусловлено особенностями использованных методов. Однако их различие не должно превышать определённого значения; этим тестом производится перекрёстная проверка правильности методов трассировки:

```
BMCRT mean luminance = 156.5
PT mean luminance    = 152.7
Relative error       = 0.6103%
RMS                  = 28.26%
-----------------------------
Status: passed
```

Следующий пример показывает методику одновременного тестирования нескольких функций программного продукта за один шаг (рис. 7). В специально созданной тестовой сцене проверяется правильность использования сложных атрибутов поверхностей, выраженных двунаправленной функцией отражения и преломления света, в различных методах трассировки лучей. Для анализа правильности результата, как и прежде, используется сравнение вновь

полученного изображения с эталонным, полученным предыдущей версией программы.

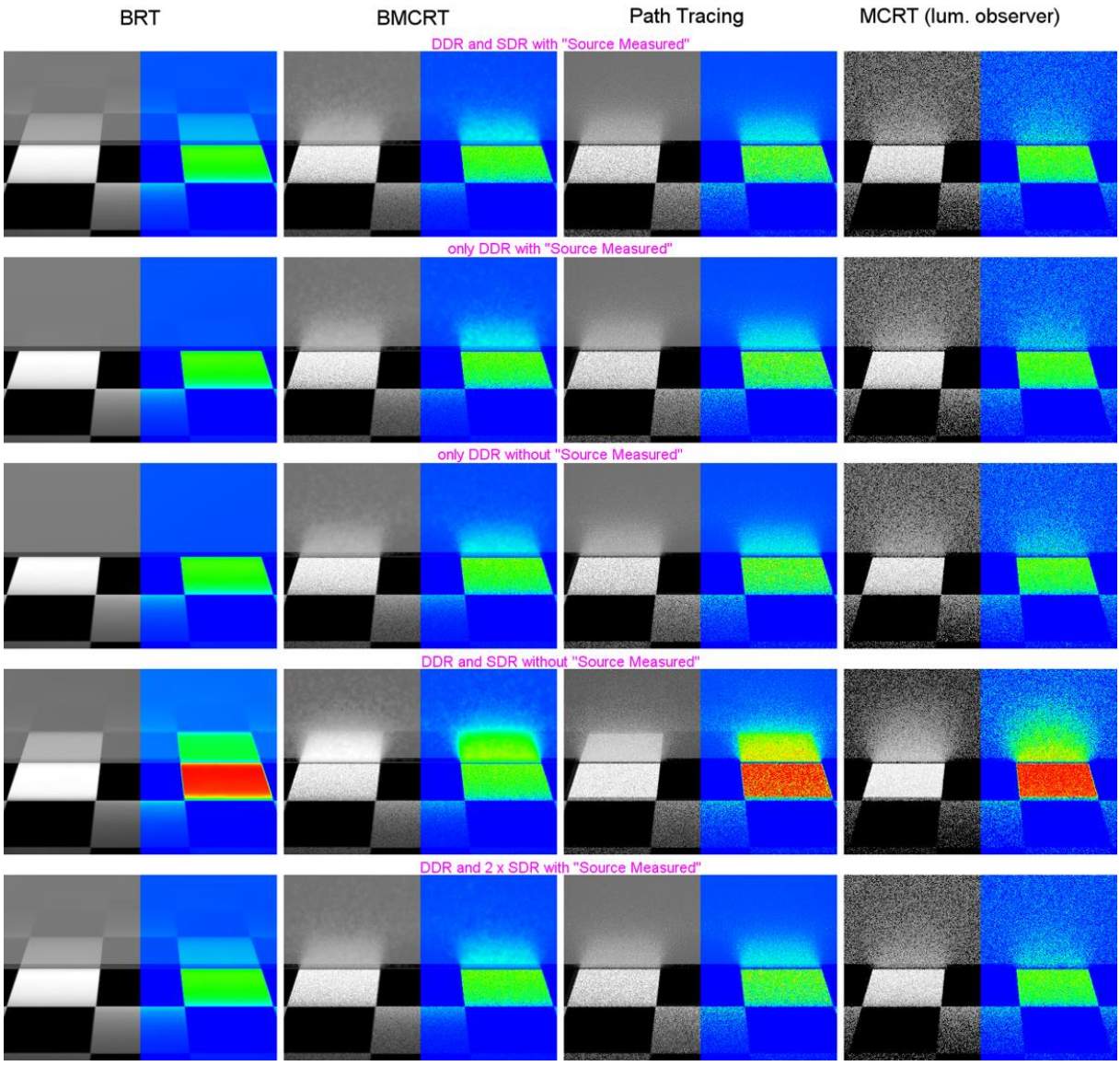


Рис. 7. Проверка корректности использования атрибутов поверхностей

И последний пример теста, сравнивающего изображение, выдаваемое тестируемой версией программы, с эталонным изображением, полученным ранее. Тестируется подсистема расчёта собственной яркости объектов сцены. Невооружённым глазом разница незаметна: нельзя сказать, являются ли изображения на рис. 8 и рис. 9 одинаковыми или нет. Для анализа используется программа, вычисляющая разницу между двумя изображениями и выдающая эту разницу в абсолютном виде и в виде изображения, удобном для восприятия (рис. 10).

Рис. 8. Эталонное изображение

Рис. 9. Изображение, полученное от тестируемой программы

Рис. 10. Разность изображений

Более яркие участки в этом разностном изображении соответствуют большей разнице в абсолютных значениях соответствующих пикселей исходных изображений. Абсолютное значение максимальной разницы между эталонным и реальным изображениями в данном случае составило 1.16%.

Необходимо отметить, что неудачное завершение теста возможно не только при ошибке в программе, но и при изменении функциональности. Кроме того, исправление одной ошибки может повлиять на результат сразу нескольких тестов. В каждом таком случае проводится анализ результатов теста, по итогам которого принимается решение о доработке теста (например, изменении эталонного изображения).

После серьёзных изменений расположения элементов пользовательского интерфейса в окнах программы интерактивный тест также может завершиться неудачно – будет выдана диагностика «невозможно найти требуемый элемент интерфейса». В этом случае необходимо изменить участок теста с учётом изменённого расположения элемента; обычно это занимает несколько минут. Хотя в большинстве случаев такие изменения не требуются – интерактивные тесты создаются максимально гибко и обычно являются устойчивыми к перемещению и добавлению элементов интерфейса в пределах диалогового окна.

## Заключение

Описанный подход позволяет предотвратить повторное появление в программных продуктах однажды уже исправленных ошибок. Очевидно, что помимо непосредственного предотвращения повторных ошибок, такая методика тестирования помогает заблаговременно выявить также и новые ошибки, которые оказывают влияние на результат, проверяемый имеющимися тестами. Это обусловлено тем, что в формировании таких результатов, как правило, используется работа сразу нескольких программных компонент и наличие ошибки в любой из них может повлиять на конечный результат расчёта.

Автоматизация такого регрессионного тестирования позволяет значительно сократить время, затрачиваемое на тестирование программных продуктов. В то время как человеку перед каждым тестом необходимо прочитать и вспомнить, что и как необходимо проверить в данном тесте, автоматическая система способна выполнять тесты один за другим, без остановок.

Описанная автоматическая система регрессионного тестирования используется при разработке и поддержке программного комплекса оптического моделирования Lumicept [9], приводя к существенной экономии времени и усилий разработчиков. За два года был создан набор из 200 тестов. Автоматическая проверка этого набора занимает около трёх часов, в то время как проверка такого объёма тестов человеком требует около двух недель. В случае необходимости время, затрачиваемое на тестирование, можно

дополнительно уменьшить путём использования для тестирования одновременно нескольких компьютеров (каждый выполняет свой набор тестов), в то время как человек способен в один момент времени обрабатывать только один тест.

## Список литературы


1. Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ, 3-е издание - М.: "Диалектика", 2012.
2. Дерябин Н.Б., Жданов Д.Д., Соколов В.Г. Внедрение языка сценариев в программные комплексы оптического моделирования // Программирование, 2017, № 1, с. 40-53. English translation: Deryabin N.B., Zhdanov D.D., Sokolov V.G. Embedding the Script Language into Optical Simulation Software // Programming and Computer Software, 2017, Vol. 43, №. 1, pp. 13-23. DOI: 10.1134/S0361768817010029
3. Test Automation FX http://www.testautomationfx.com (дата обращения: 09.10.2018)
4. TestComplete https://smartbear.com/product/testcomplete/overview (дата обращения: 09.10.2018)
5. Squish http://www.froglogic.com/squish (дата обращения: 09.10.2018)
6. Sikuli http://www.sikuli.org (дата обращения: 09.10.2018)
7. Atoma http://www.getautoma.com (дата обращения: 09.10.2018)
8. AutoIt http://www.autoitscript.com (дата обращения: 09.10.2018)
9. Жданов Д.Д., Потемин И.С., Галактионов В.А., Барладян Б.Х., Востряков К.А., Шапиро Л.З. Спектральная трассировка лучей в задачах построения фотореалистичных изображений // "Программирование", 2011, № 5, с. 13-26.
10. Денисов Е.Ю., Волобой А.Г., Калугина И.А. Автоматическое тестирование графического интерфейса интерактивных программных комплексов // Новые информационные технологии в автоматизированных системах: материалы двадцать первого научно-практического семинара - М.: ИПМ им. М.В.Келдыша, 20 апреля 2018, с. 83-88.
11. Денисов Е.Ю., Волобой А.Г., Калугина И.А. Автоматическое регрессионное тестирование программных комплексов // Новые информационные технологии в автоматизированных системах: материалы двадцать первого научно-практического семинара. — М.: ИПМ им. М.В.Келдыша, 20 апреля 2018, с. 89-94.

12. Волобой А.Г., Денисов Е.Ю., Барладян Б.Х. Тестирование систем моделирования освещенности и синтеза реалистичных изображений // Программирование, № 4, 2014, с.13-22. *English translation:* A. G. Voloboi, E. Yu. Denisov, B. Kh. Barladyan. Testing of Systems for Illumination Simulation and Synthesis of Realistic Images // Programming and Computer Software, 2014, Vol. 40, № 4, pp. 166–173. DOI:10.1134/S0361768814040094