Автоматическое регрессионное тестирование программных комплексов

Денисов Е.Ю., Волобой А.Г., Калугина И.А., ИПМ им. М.В. Келдыша РАН
eed@spp.keldysh.ru, voloboy@gin.keldysh.ru, qik@gin.keldysh.ru

Аннотация

В статье описана технология автоматического регрессионного тестирования, применяемая при разработке и сопровождении в пределах жизненного цикла современных программных комплексов компьютерной графики и оптического моделирования.

1 Введение

Программные комплексы оптического моделирования, разрабатываемые нашим коллективом, имеют следующие особенности:

- Разнородные программные комплексы используют общие компоненты;
- Частые выпуски новых версий;
- Ограниченные ресурсы и время.

Сложность современных программных комплексов вообще, и наших продуктов в частности, настолько высока, что классические технологии тестирования [Майерс, Баджетт, Сандлер, 2012] требуют много времени и значительных человеческих ресурсов, которых обычно не хватает. В этих условиях большую роль в повышении эффективности разработки играет автоматизация тестирования программ.

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 причиной 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 исправленная ошибка. 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.

2 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.

3 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.

3.1 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 анализа 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 пиксе-

лями (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).

3.2 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 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».

3.3 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. Для тестирования пользовательского интерфейса применяется программный пакет AutoIt [AutoIt], позволяющий имитировать воздействия пользователя на органы управления (клавиатуру, мышь) и взаимодействие пользователя с элементами пользовательского интерфейса (окна, диалоги, кнопки, переключатели, поля ввода, и т.д.). Указанные воздействия записываются в виде BASIC-подобного языка сценариев и могут быть воспроизведены в дальнейшем. С помощью команд языка сценариев AutoIt также может быть проанализировано состояние тестируемой программы: наличие на экране определённых окон и диалогов, их положение и размер; наличие, расположение, размер и состояние элементов пользовательского интерфейса; содержимое полей ввода и т.д. Кроме того, имеются также методы работы с файлами, библиотека математических операций, операции работы со строками, методы контроля выполняемых процессов, и т.д. В совокупности эти механизмы позволяют реализовать сложные сценарии работы пользователя с интерактивным приложением.

Однако, хочется отметить, что создание таких тестов довольно трудоемко, так как фактически каждый тест требует написания программы на языке сценариев AutoIt. Также, возможно, что с изменением пользовательского интерфейса будущих версий системы потребуется коррекция и текста самого теста.

4 Примеры тестирования

Ниже приведены несколько типичных результатов тестирования.

1) 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.

2) 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.

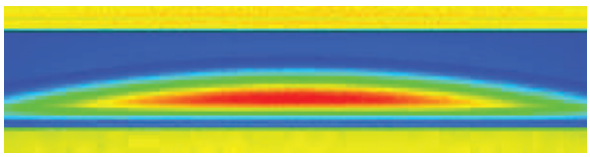

Рис. 1. BRT, эталонное изображение

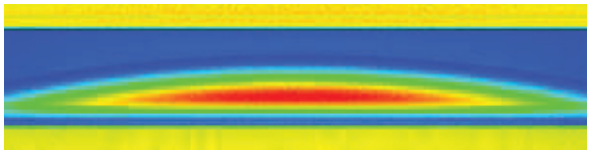

Рис. 2. BRT, проверяемое изображение

Разница между изображениями не должна превышать определённого значения:

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

3) 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
```

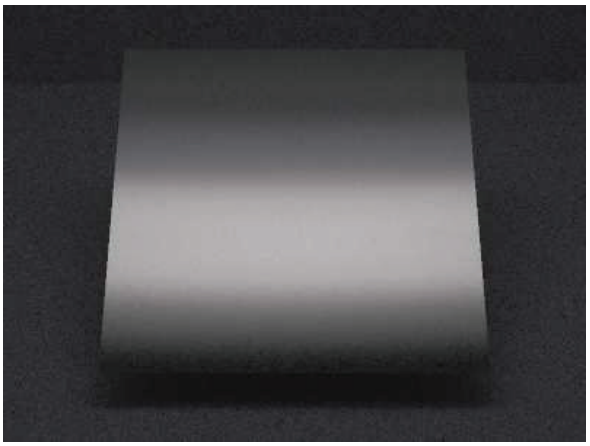

Рис. 5. Метод BMCRT

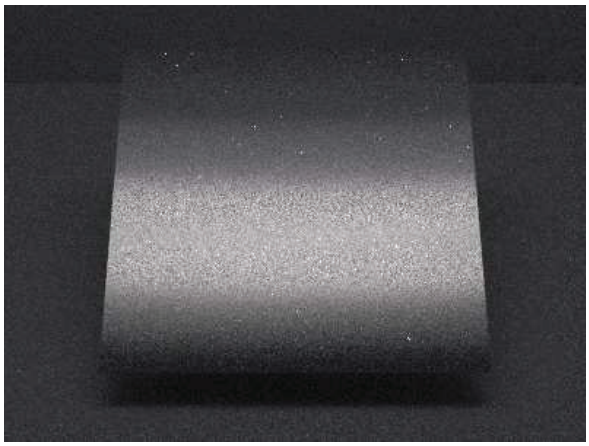

Рис. 6. Метод PT

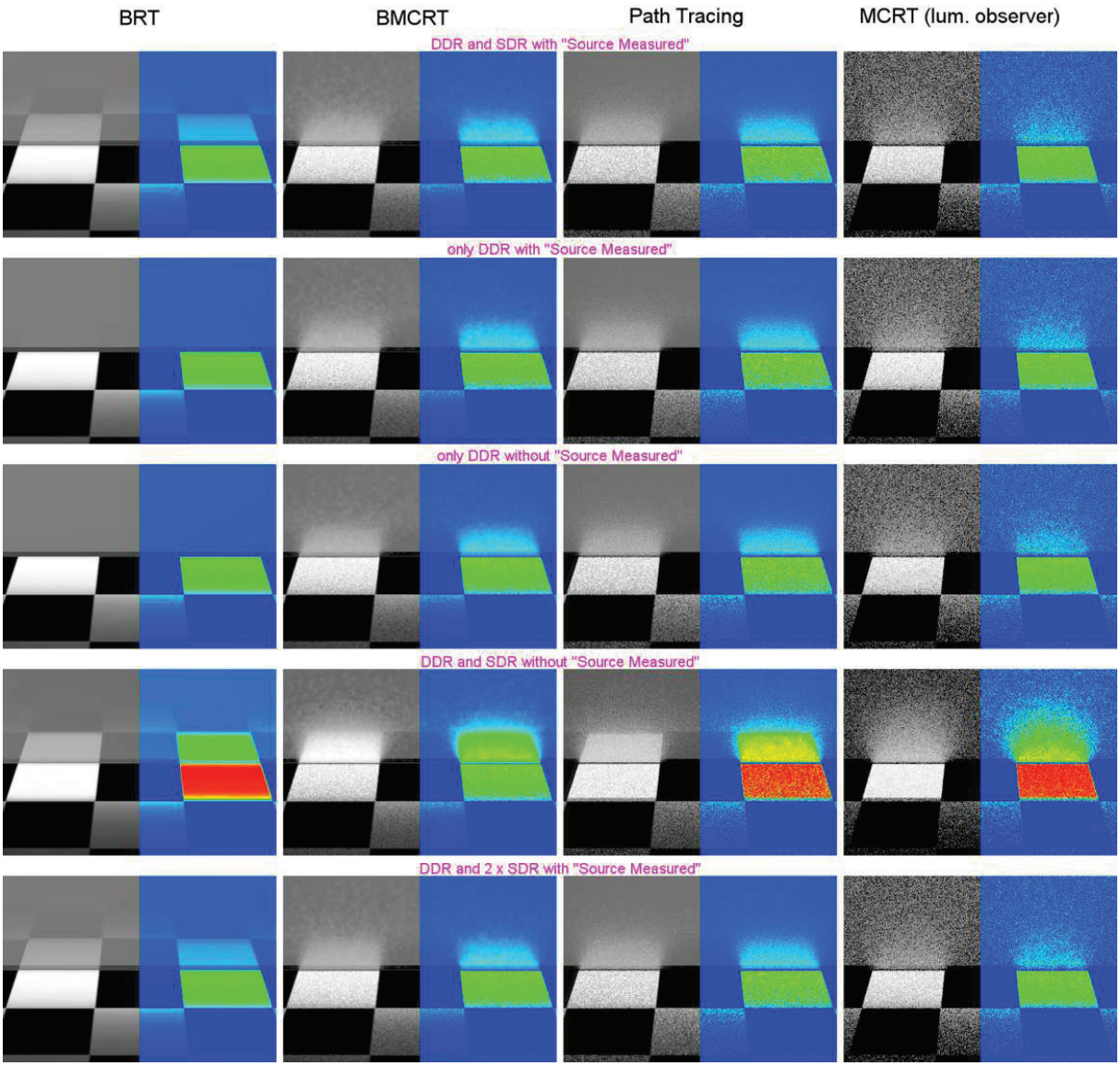

Рис. 7. Проверка корректности использования атрибутов поверхностей

4) 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. 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
```

5) И, наконец, последний пример на рис. 7 показывает методику одновременного тестирования нескольких функций программного продукта за один шаг.

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.

5 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 [Жданов, Потемин, Галактионов, Барладян, Востряков, Шапиро, 2011], приводя к существенной экономии времени и усилий разработчиков. Автоматическая проверка набора из двухсот тестов занимает около четырёх часов, в то время как проверка такого объёма тестов человеком требует от одной до 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.

Благодарности

Работа выполнена при частичной поддержке РФФИ, гранты № 16-01-00552, 18-01-00569.

Список литературы

Г. Майерс, Т. Баджетт, К. Сандлер. Искусство тестирования программ, 3-е издание - М.: "Диалектика", 2012.

AutoIt
URL: http://www.autoitscript.com/ (дата обращения: 19.02.2018)

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.