Высококачественная объемная визуализации в реальном времени

Денис Боголепов[1], Илья Бугаев[1], Дмитрий Сопин[1], Данила Ульянов[2], Вадим Турлапов[1]
[1]Нижегородский государственный университет им. Н.И. Лобачевского
[2]Нижегородский государственный технический университет им. Р.Е. Алексеева
denisbogol@gmail.com, sopindm@gmail.com, danila-ulyanov@ya.ru

Аннотация

В настоящей работе рассматривается алгоритм прямой визуализации объема, построенный на методе испускания лучей и адаптированный для программируемой графической аппаратуры. Алгоритм использует прединтегрированную классификацию, которая обеспечивает высокое качество визуализации при относительно небольших вычислительных затратах. Для работы алгоритма практически не требуется предварительных вычислений, что позволяет его встраивать в системы, допускающие модификацию передаточных функций в реальном времени. Предлагаются оптимизационные стратегии для повышения качества и скорости работы. Приводятся оценки производительности.

Ключевые слова: *direct volume rendering, ray casting, pre-integrated classification, GPU, GPGPU, OpenGL, GLSL.*

1. ВВЕДЕНИЕ

Объемная визуализация – метод формирования изображения, который для трехмерного набора данных отображает не только общий вид, но и внутреннее строение трехмерного объекта. В качестве входного набора данных часто выступает множество плоских изображений слоев, полученных с помощью компьютерной или магнитно-резонансной томографии. Обычно слои имеют равную толщину и одинаковое число пикселей на каждый слой. Таким образом, входные данные можно представить регулярной пространственной сеткой, каждому узлу которой ставится в соответствие значение некоторого скалярного поля.

Существует большое разнообразие методов визуализации объема, однако наилучшие результаты обеспечивает алгоритм трассировки лучей, который естественным образом вычисляет интеграл объемного рендеринга. Значительное влияние на результирующее изображение также оказывает метод классификации, посредством которого конкретным значениям скалярного поля ставится в соответствие набор оптических свойств. Высокое качество обеспечивает так называемая прединтегрированная классификация, которая была предложена в работе [1]. Однако доступная на тот момент аппаратура не поддерживала программируемый графический конвейер, поэтому метод был реализован с помощью техники слоев [2]. С ростом возможностей программируемых графических процессоров появились интерактивные реализации прямого объемного рендеринга на основе испускания лучей [3]. В публикации [4] авторы представили программный каркас системы объемной визуализации с использованием прединтегрированной классификации и такими эффектами, как объемные тени и освещение. Исследованию различных методов интегрирования посвящена работа [5], в которой авторы предложили использовать интерполяцию второго порядка для аппроксимации скалярного поля в пределах интервала дискретизации. Наряду с этим следует отметить работы, направленные на повышение наглядности визуализации и упрощение анализа изображения. В частности, рассматривалась визуализация в широком диапазоне (HDR), объемные тени, различные модели освещения [6]–[7].

Настоящая работа посвящена реализации высококачественного объемного рендеринга, в основе которого лежит метод испускания лучей. Рассматриваются особенности реализации прединтегрированной классификации на графическом процессоре и исследуются ее преимущества по сравнению с традиционной пост-классификацией. Предлагается ряд оптимизационных стратегий, направленных на улучшение качества изображения и повышение скорости работы. Приводится сравнительный анализ различных вариантов реализации метода.

2. ОСНОВНЫЕ МОДЕЛИ И МЕТОДЫ

2.1 Интеграл объемной визуализации

Предположим, что луч $\mathbf{x}(t)$ параметризован расстоянием t от объектива виртуальной камеры, а излучаемая яркость intensity($\mathbf{x}$) и коэффициент затухания extinction($\mathbf{x}$) могут быть вычислены в любой точке $\mathbf{x}$ пространства. Интеграл объемной визуализации [1] описывает процесс накопления яркости вдоль луча с учетом ее затухания в зависимости от расстояния до точки наблюдения:

$$I = \int_0^D \text{intensity}(\mathbf{x}(t)) \times \text{extinction}(\mathbf{x}(t)) \times \exp\left(-\int_0^t \text{extinction}(\mathbf{x}(t'))\mathrm{d}t' \right) \mathrm{d}t \quad (1)$$

Здесь через D обозначено расстояние, на которое луч проникает в объем (в большинстве случаев определяется временем выхода из ограничивающей оболочки).

2.2 Скалярное поле

На практике скалярное поле $s(\mathbf{x})$ часто задается регулярной трехмерной сеткой, каждому узлу $\mathbf{v}_i$ которой ставится в соответствие значение поля s_i. Для вычисления поля в произвольной точке пространства выполняется интерполяция. Порядок данной интерполяции существенно влияет на качество генерируемого изображения. Современные графические процессоры аппаратно поддерживают трилинейную фильтрацию. Лучшие результаты обеспечивает трикубическая фильтрация, которая может быть реализована через 8 операций трилинейной фильтрации [9].

2.3 Передаточные функции

Преобразование скалярного поля $s(\mathbf{x})$ в яркость излучаемого света и коэффициент затухания называется классификацией. Данное преобразование выполняется путем задания передаточных функций для вычисления излучаемой яркости $L(s)$ и коэффициента затухания $\tau(s)$. При наличии передаточных функций интеграл (1) записывается в виде:

$$I = \int_0^D L(s(\mathbf{x}(t))) \times \tau(s(\mathbf{x}(t))) \times \exp\left(-\int_0^t \tau(s(\mathbf{x}(t'))) \mathrm{d}t' \right) \mathrm{d}t \quad (2)$$

В ряде случаев для вычисления яркости $L(s)$ в конкретной точке луча $\mathbf{x}(t)$ могут использоваться алгоритмы затенения, основанные на различных моделях освещения (таких как модели Блинна-Фонга или Эми Гуч).

2.4 Пред- и пост-классификация

В зависимости от способа вычисления передаточных функций выделяют различные типы классификации. В случае пост-классификации передаточные функции применяются после интерполяции скалярного поля $s(\mathbf{x})$. В случае предклассификации излучаемая яркость и коэффициент затухания вычисляются на этапе препроцессирования для каждого узла сетки $\mathbf{v}_i$ и используются для интерполяции в произвольную точку пространства. На практике данный подход практически не применяется.

2.5 Численное интегрирование

В большинстве случаев для оценки интеграла (2) используется простой метод прямоугольников. Будем считать, что отрезок интегрирования разбивается на n элементарных отрезков равной длины $d = D / n$. Тогда справедливы следующие квадратурные формулы:

$$\begin{aligned}\exp\left(-\int_0^t \tau(s(\mathbf{x}(t')))\mathrm{d}t'\right) &\approx \exp\left(-\sum_{i=0}^{t/d}\tau(s(\mathbf{x}(id)))d\right)\\ &= \prod_{i=0}^{t/d}\exp(-\tau(s(\mathbf{x}(id)))d) = \prod_{i=0}^{t/d}(1-\alpha_i)\end{aligned} \tag{3}$$

Здесь через α_i обозначена непрозрачность i-ого сегмента луча. Разложение экспоненты в ряд Тейлора позволяет дополнительно упростить выражение:

$$\alpha_i = 1-\exp(-\tau(s(\mathbf{x}(id)))d) \approx \tau(s(\mathbf{x}(id)))d \tag{4}$$

Данная формула предпочтительна в задачах визуализации реального времени, поскольку вычисление экспоненциальной функции существенно снижает производительность. Аналогичную оценку можно получить для яркости, излучаемой i-ым сегментом луча:

$$C_i \approx L(s(\mathbf{x}(id)))\tau(s(\mathbf{x}(id)))d \tag{5}$$

В результате, получаем следующую аппроксимацию для интеграла (2):

$$I \approx \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1}(1-\alpha_j) \tag{6}$$

Полученная аппроксимация сходится при $d \to 0$. На практике для точной оценки непрерывного подынтегрального выражения необходимо рационально выбрать шаг дискретизации. Для понимания сути проблемы уместно обратиться к теореме отсчетов (Котельникова), согласно которой корректное восстановление аналогового сигнала возможно только по дискретным отсчетам, взятым с частотой строго большей частоты Найквиста (удвоенной верхней частоты сигнала). Скачкообразный характер передаточных функций приводит к значительному повышению частоты дискретизации, поскольку частота Найквиста у подынтегрального выражения в худшем случае оценивается произведением данных частот у поля $s(\mathbf{x})$ и передаточных функций $L(s)$ и $\tau(s)$. В итоге корректное вычисление интеграла может потребовать сотен тысяч интервалов дискретизации для каждого луча. Для работы алгоритмов визуализации в реальном времени число интервалов ограничивается несколькими тысячами, что приводит к различным артефактам в результирующем изображении.

2.6 Прединтегрированная классификация

Эффективным способом борьбы с высокими частотами Найквиста является прединтегрированная классификация, которая подробно рассматривается в работе [1]. Основная идея сводится к декомпозиции вычислений на интегрирование скалярного поля $s(\mathbf{x})$ и интегрирование передаточных функций $L(s)$ и $\tau(s)$. Для этого на этапе препроцессирования генерируются таблицы, с помощью которых можно получить яркость и непрозрачность произвольного сегмента луча во время визуализации. В общем случае для обращения к таблицам используется три аргумента: значение поля в начальной точке сегмента $s_f = s(\mathbf{x}(id))$, значение поля в конечной точке сегмента $s_b = s(\mathbf{x}(id + d))$ и длина сегмента d. Если же лучи разбиваются на сегменты равной длины, то число аргументов сокращается до двух. Непрозрачность i-ого сегмента выражается формулой:

$$\begin{aligned}\alpha_i(s_f, s_b, d) &= 1-\exp\left(-\int_{id}^{id+d}\tau(s(\mathbf{x}(t')))\mathrm{d}t'\right)\\ &\approx 1-\exp\left(-\int_0^1 \tau((1-\lambda)s_f+\lambda s_b)d\,\mathrm{d}\lambda\right)\\ &\approx \int_0^1 \tau((1-\lambda)s_f+\lambda s_b)d\,\mathrm{d}\lambda\end{aligned} \tag{7}$$

Аналогично оценивается яркость i-ого сегмента луча:

$$\begin{aligned}C_i(s_f, s_b, d) \approx \int_0^1 &L((1-\lambda)s_f+\lambda s_b)\times\tau((1-\lambda)s_f+\lambda s_b)\\ &\times\exp\left(-\int_0^\lambda \tau((1-\lambda')s_f+\lambda' s_b)d\,\mathrm{d}\lambda'\right)d\,\mathrm{d}\lambda\end{aligned} \tag{8}$$

Таким образом, прединтегрированная классификация позволяет не увеличивать частоту дискретизации для скачкообразных передаточных функций. В итоге имеется потенциал для улучшения качества визуализации и уменьшения числа интервалов дискретизации на этапе визуализации объема.

2.7 Упрощенное прединтегрирование

Основной недостаток прединтегрированной классификации связан с необходимостью вычисления таблиц, с помощью которых каждому набору параметров s_f, s_b и d ставится в соответствие яркость сегмента и его непрозрачность. Указанные таблицы зависят от передаточных функций, поэтому требуют повторного построения при их модификации. Данное построение может быть эффективно выполнено на графическом процессоре, что позволит модифицировать передаточные функции в реальном времени.

Альтернативным вариантом является аппроксимированная прединтегрированная классификация [1], которая без видимого снижения качества позволяет ограничиться быстрым предварительным вычислением двух первообразных функций:

$$T(s) = \int_0^s \tau(\lambda)\mathrm{d}\lambda \tag{9}$$

$$K(s) = \int_0^s L(\lambda)\tau(\lambda)\mathrm{d}\lambda \tag{10}$$

С использованием данных функций формулы (7) и (8) могут быть преобразованы следующим образом:

$$\alpha_i(s_f, s_b, d) \approx \frac{d}{s_b - s_f}\left[T(s_b) - T(s_f)\right] \quad (11)$$

$$C_i(s_f, s_b, d) \approx \frac{d}{s_b - s_f}\left[K(s_b) - K(s_f)\right] \quad (12)$$

Выражение (12) предполагает отсутствие затухания в пределах одного сегмента луча, что оправданно при малом интервале дискретизации.

2.8 Стратегии вычисления цвета

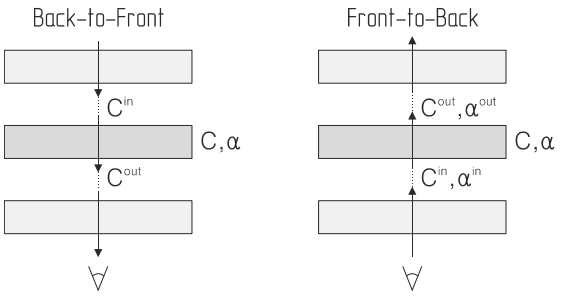

Рис. 1: Стратегии накопления цвета.

На основе аппроксимации (6) строится алгоритм *обратного* (*back-to-front*) вычисления цвета пикселя:

$$C^{out} \leftarrow C + (1 - \alpha)C^{in} \quad (13)$$

Алгоритм *прямого* (*front-to-back*) вычисления цвета также возможен, однако требует дополнительных ресурсов для поддержки коэффициента непрозрачности:

$$\begin{aligned} C^{out} &\leftarrow C^{in} + (1 - \alpha^{in})C \\ \alpha^{out} &\leftarrow \alpha^{in} + (1 - \alpha^{in})\alpha \end{aligned} \quad (14)$$

Несмотря на использование дополнительных регистров и большее число операций в цикле трассировки, алгоритм прямого вычисления цвета часто более предпочтителен за счет возможности ранней остановки луча (early ray termination). В настоящей работе реализованы обе стратегии трассировки.

2.9 Стратегии генерации лучей

В типовой реализации метода объемного рендеринга цвет каждого пикселя вычисляется с помощью одного первичного луча, для оценки яркости которого используется S интервалов дискретизации. В качестве альтернативного подхода можно предложить генерацию R лучей на пиксель, начальные точки которых последовательно смещаются в пределах интервала дискретизации, а число интервалов сокращается до S / R. Очевидно, что вычислительные затраты в обоих случаях одинаковы, однако второй подход имеет ряд преимуществ.

Во-первых, программный код имеет большую степень параллелизма, поскольку лучи в пределах пикселя обрабатываются независимо. Шейдерный компилятор получает возможность эффективного распараллеливания инструкций, что ведет к более полному использованию ресурсов ГПУ.

Во-вторых, данный подход позволяет реализовать практически бесплатное полноэкранное сглаживание (FSAA). Для этого каждый из R лучей необходимо дополнительно смещать в пределах пикселя экранной плоскости. При этом эффективно устраняется не только эффект ступенчатости изображения, но и мелкие (сопоставимые с размерами пикселя) артефакты визуализации.

В-третьих, лучи в пределах пикселя характеризуются высокой пространственной когерентностью, что позволяет дополнительно повысить быстродействие. Трассировка первого луча через центр пикселя в качестве побочного результата дает оценку расстояния, которое луч преодолел в пустом пространстве. Данная информация позволяет трассировать остальные лучи сразу со значимой области исследуемого объема. В ряде случаев такая оптимизация увеличивает скорость работы в несколько раз, поскольку значительно сокращается число наиболее трудоемких операций – выборок из трехмерной текстуры скалярного поля (оценки производительности приводятся далее).

3. РЕАЛИЗАЦИЯ АЛГОРИТМА НА ГПУ

Несмотря на широкую доступность таких универсальных инструментов программирования ГПУ как NVIDIA CUDA и OpenCL, в настоящей работе для этой цели использовался интерфейс OpenGL и его шейдерный язык GLSL (OpenGL Shading Language). Богатый набор встроенных типов данных и широкие функциональные возможности позволяют в естественном виде записывать алгоритм испускания лучей. К достоинствам подхода следует также отнести поддержку графических ускорителей от различных производителей. Интерфейс OpenGL и его язык шейдеров являются межплатформенным стандартом, что позволяет использовать единый программный код для поддержки самых различных платформ.

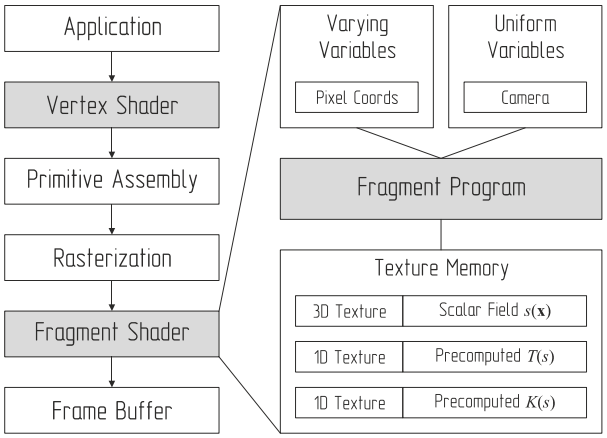

Рис. 2: Схема работы алгоритма визуализации.

Существует стандартный подход к отображению алгоритма испускания лучей на программируемый графический конвейер. Для инициирования вычислений следует установить параллельную проекцию и нарисовать прямоугольник, заполняющий всю область видимости в окне размера $N \times M$ пикселей. На этапе растеризации данного прямоугольника будет сгенерировано ровно $N \times M$ фрагментов, соответствующих пикселям в буфере кадра. Каждый сгенерированный фрагмент обрабатывается фрагментным шейдером, на вход которому передаются координаты пикселя и данные для визуализации. Располагая всей необходимой информацией, фрагментный шейдер генерирует первичный луч (несколько лучей) и трассирует его через объем согласно изложенному выше алгоритму. Шейдерный язык GLSL поддерживает директивы препроцессора, с помощью которых можно быстро переключаться между блоками кода, реализующими различные методы визуализации. В тестовой версии программы пользователь может выбирать между пост-классификацией и прединтегрированной классификацией, а также между прямым и обратным алгоритмом прохода по лучу.

4. АНАЛИЗ РЕЗУЛЬТАТОВ

Для оценки качества и скорости работы рассмотренных алгоритмов использовались широко известные данные для объемной визуализации [10]. Расчеты производились на графическом процессоре начального уровня NVIDIA GeForce 560 1 Гб под управлением ОС Ubuntu Linux 10.10 и видеодрайвера NVIDIA Linux Graphics Driver 275.36. Визуализация выполнялась в режимах пост-классификации и прединтегрированной классификации, при этом сравнивались прямой и обратный алгоритм вычисления цвета. Для обработки каждого пикселя генерировалось 5 лучей, каждый из которых разбивался не более чем на 120 интервалов. Таким образом, общее число дискретных отсчетов на пиксель не превышало 600, что является скромным показателем для современной графической аппаратуры. Для каждого режима визуализации выполнялось два замера: без использования и с использованием когерентности субпиксельных лучей. В следующей таблице представлены характеристики входных данных и результаты эксперимента.

Таблица 1: Результаты замера производительности (кадров / секунду, окно 512 × 512 пикселей).

Description	Dimensions Bits	Post-classification		Pre-integrated classification	
		Front-to-Back	Back-to-Front	Front-to-Back	Back-to-Front
Bucky Ball	32 × 32 × 32 8 bit	120 / 142	82 / 119	150 / 152	75 / 140
Daisy Pollen Grain	192 × 180 × 168 8 bit	59 / 100	58 / 98	66 / 108	66 / 112
Engine Block	256 × 256 × 256 8 bit	71 / 118	67 / 130	74 / 110	71 / 130
Bonsai Tree	512 × 512 × 154 8 bit	61 / 128	63 / 89	66 / 114	51 / 90
Stanford Bunny	512 × 512 × 361 16 bit	36 / 92	38 / 79	38 / 47	34 / 44
Orange	256 × 256 × 64 8 bit	61 / 86	75 / 98	84 / 100	60 / 120

Экспериментальные данные показывают, что прединтегрированная классификация практически не вносит дополнительных затрат на этапе визуализации по сравнению с постклассификацией. В данном случае скорость работы ограничивается производительностью текстурных модулей, которые интенсивно используются для выборки и фильтрации значений скалярного поля. Во всех режимах работы число данных операций остается неизменным.

Прединтегрированная классификация (рис. 3) обеспечивает значительно более высокое качество изображения при использовании всего 600 интервалов дискретизации на каждый пиксель. Таким образом, в особо трудных случаях (при работе с высокочастотными передаточными функциями и скалярным полем) остается значительный ресурс повышения качества за счет увеличения числа интервалов (вплоть до нескольких раз при сохранении реального времени).

Без использования когерентности субпиксельных лучей наилучшие результаты дает метод прямого накопления цвета с прединтегрированной классификацией. Очевидно, что данный режим работы обеспечивает наилучшие условия для ранней остановки луча. Использование когерентности для отсечения пустого пространства позволяет значительно повысить производительность. Величина ускорения существенно зависит от входных данных и передаточных функций, однако в ряде случаев наблюдается прирост до двух и более раз. Кроме того, использование когерентности зачастую выводит метод обратного накопления цвета в число наиболее эффективных. Поскольку выбрать единственный оптимальный подход не представляется возможным, в прикладных программах целесообразно поддерживать различные режимы визуализации.

5. ЗАКЛЮЧЕНИЕ

В настоящей работе был реализован алгоритм прямой объемной визуализации для программируемой графической аппаратуры на базе метода испускания лучей. Алгоритм использует прединтегрированную классификацию, которая обеспечивает высокую скорость работы и качество визуализации даже для нелинейных передаточных функций с высокими частотами. Для работы алгоритма практически не требуется предварительных вычислений, что позволяет модифицировать передаточные функции в реальном времени. Предложены подходы экономичного полноэкранного сглаживания и отсечения пустого пространства от начальной точки луча до значимой области объема. Реализована тестовая версия алгоритма на базе интерфейса OpenGL и шейдерного языка GLSL, которые обеспечивают высокую производительность на различных программных и аппаратных платформах.

6. ЛИТЕРАТУРА

[1] K. Engel, M. Kraus, T. Ertl. *High-quality pre-integrated volume rendering using hardware-accelerated pixel shading* // In Proc. ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware (HWWS '01), 9-16.

[2] J. Edward Swan, Roni Yagel. *Slice-Based Volume Rendering* // Ohio State University Technical Report OSU-ACCAD-1/93-TR1, January 1993.

[3] J. Krüger, R. Westermann. *Acceleration Techniques for GPU-based Volume Rendering* // In Proc. 14th IEEE Visualization 2003 (VIS'03), 287-292.

[4] J. E. Vollrath, D. Weiskopf, and T. Ertl. *A generic software framework for the gpu volume rendering pipeline* // In Proc. Vision, Modeling, and Visualization 2005, 391-398.

[5] J.-F. El Hajjar, S. Marchesin, J.-M. Dischler, C. Mongenet. *Second Order Pre-Integrated Volume Rendering* // In Proc. Visualization Symposium 2008 (PacificVIS '08), 9-16.

[6] A. Guetat, A. Ancel, S. Marchesin, J.-M. Dischler. *Pre-Integrated Volume Rendering with Non-Linear Gradient Interpolation* // IEEE Trans Vis Comput Graph, 16 (6), 2010, 1487-94.

[7] D. S. Ebert and P. Rheingans. *Volume illustration: Non-photorealistic rendering of volume models* // In Proc. Visualization '00 (VIS '00), 195-202.

[8] Nelson Max, Pat Hanrahan, and Roger Crawfis. *Area and volume coherence for efficient visualization of 3D scalar functions* // In Proceedings of the 1990 workshop on Volume visualization (VVS '90), 27-33.

[9] Christian Sigg, Markus Hadwiger. *Fast Third-Order Texture Filtering* // In GPU Gems 2, Volume 2, 313-318.

[10] The Volume Library by Stefan Roettger: http://www9.informatik.uni-erlangen.de/External/vollib

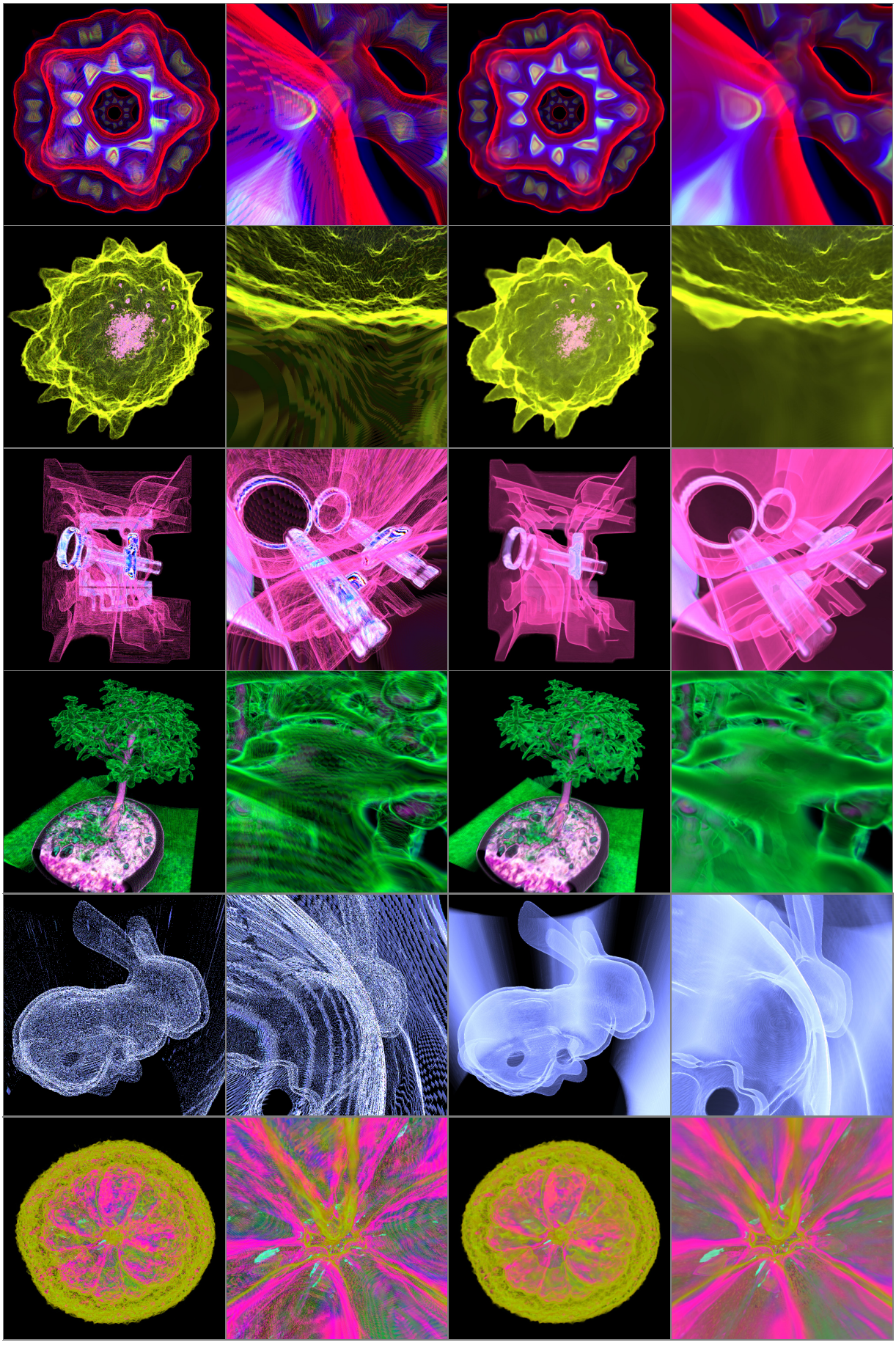

Рис. 3: Пост-классификация (*первые два столбца*) и прединтегрированная классификация (*последние два столбца*). Входные данные (*сверху вниз*): Bucky Ball, Daisy Pollen Grain, Engine Block, Bonsai Tree, Stanford Bunny, Orange.