

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>5</b>
1.1 Трассировка лучей . . . . .	5
1.1.1 Классическая трассировка лучей . . . . .	5
1.1.2 Трассировка лучей методом Монте-Карло . . . . .	6
1.2 Квантовая обработка изображений . . . . .	7
1.2.1 Использование избыточной выборки . . . . .	7
1.3 Основные положения квантовых вычислений . . . . .	9
1.3.1 Квантовый бит . . . . .	9
1.3.2 Квантовый регистр . . . . .	10
1.4 Синтез изображения в квантовом представлении . . . . .	12
1.4.1 Квантовый пиксельный шейдер . . . . .	12
1.4.2 Квантовая фаза . . . . .	13
1.4.3 Усиление комплексной амплитуды . . . . .	14
1.4.4 Квантовая фазовая логика . . . . .	16
1.4.5 Квантовое преобразование Фурье . . . . .	17
1.5 Квантовая избыточная выборка . . . . .	19
1.5.1 Принцип работы . . . . .	20
1.5.2 Поисковая таблица . . . . .	20
1.5.3 Карта достоверности . . . . .	22
1.6 Колоризация изображения . . . . .	23
<b>2 Конструкторская часть</b>	<b>25</b>
2.1 Схема квантового компьютера . . . . .	25
2.1.1 Квантовая коррекция ошибок . . . . .	25
2.1.2 Пользовательский интерфейс квантового компьютера	25
2.1.3 Квантовая операционная система . . . . .	26
2.2 Эмуляция квантовых вычислений . . . . .	27
2.2.1 Квантовое превосходство . . . . .	28
2.3 Структуры данных для квантового алгоритма избыточной выборки . . . . .	28
2.3.1 Квантовая поисковая таблица . . . . .	28

2.3.2	Квантовая карта достоверности . . . . .	29
2.4	Алгоритм квантовой избыточной выборки . . . . .	29
2.4.1	Формирование квантовой поисковой таблицы . . . . .	30
2.4.2	Формирование квантовой карты достоверности . . . . .	30
<b>3</b>	<b>Технологическая часть</b>	<b>31</b>
3.1	Средства реализации . . . . .	31
3.2	Детали реализации . . . . .	32
<b>4</b>	<b>Исследовательская часть</b>	<b>37</b>
4.1	Результаты работы программного обеспечения . . . . .	37
4.2	Постановка эксперимента . . . . .	39
4.2.1	Цель эксперимента . . . . .	39
4.2.2	Сравнение уровня зашумленности . . . . .	39
4.2.3	Сравнение характера шума . . . . .	39
4.2.4	Сравнение вычислительной сложности алгоритмов . . . . .	40
	<b>Заключение</b>	<b>43</b>
	<b>Литература</b>	<b>44</b>

# Введение

Трассировка лучей – метод геометрической оптики – исследование оптических систем путём отслеживания взаимодействия отдельных лучей с поверхностями. Специальный алгоритм отслеживает путь луча, начиная от объекта освещения до объектов, расположенных на сцене. Далее, алгоритм создает симуляцию взаимодействия с объектами: отражение, преломление и так далее. Полученная информация используется для определения цвета каждого пикселя в итоговом изображении.

Качество результирующего изображения напрямую зависит от количества испускаемых лучей, а увеличение количества лучей, в свою очередь, требует повышения затрат вычислительных ресурсов. Трассировка лучей используется в киноиндустрии [1], что мотивирует искать способы снижения затрат на синтез изображения с помощью трассировки лучей. Так, например, для высокобюджетного фильма с 24 кадрами в секунду, на рендеринг одного кадра уходит до двух часов [1].

Квантовые вычисления – это альтернатива классическим алгоритмам, основанная на процессах квантовой физики, которая гласит, что без взаимодействия с другими частицами (то есть до момента измерения), электрон не размещен в однозначных координатах, а одновременно расположен в каждой точке орбиты. Область, в которой расположен электрон, называется электронным облаком, а феномен нахождения в каждой точке орбиты – суперпозиция. В ходе опыта Юнга, эксперимента с двумя щелями один электрон проходит одновременно через обе щели, интерферируя при этом с самим собой. Только при измерении эта неопределенность схлопывается и координаты электрона становятся однозначными.

Случаи, когда квантовый алгоритм работает хотя бы немного быстрее чем его классическая версия, редки [2]. Однако, это не умаляет значения квантовых вычислений, потому что они способны ускорить выполнение задач переборного типа.

Цель работы – реализовать ПО, в котором реализованы квантовые алгоритмы, которые в дальнейшем возможно применить в алгоритме трассировки лучей с целью его улучшения.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- проанализировать алгоритм трассировки лучей, чтобы понять какую часть вычислений стоит заменить на квантовые;
- проанализировать и сконструировать выбрать квантовые алгоритмы и структуры данных, которые возможно использовать в алгоритме трассировки;
- реализовать выбранные квантовые алгоритмы.
- провести сравнение рассматриваемых алгоритмов с использованием квантовых и традиционных вычислений.

# 1 Аналитическая часть

## 1.1 Трассировка лучей

### 1.1.1 Классическая трассировка лучей

Трассировка лучей (англ. ray tracing) – метод синтеза компьютерных изображений, который позволяет добиться повышения качества синтезируемого изображения за счёт повышения затрат вычислительных ресурсов. Для каждого пикселя в итоговом изображении математический луч проецируется в трехмерном пространстве из камеры через пиксель по направлению к сцене. В классическом варианте луч сталкивается с объектом сцены, и, не учитывая отражение и прозрачность, с помощью цвета объекта, определяется цвет соответствующего пикселя в синтезируемом изображении. На рисунке 1.1 представлено схематическое представление построения изображения методом трассировки лучей.

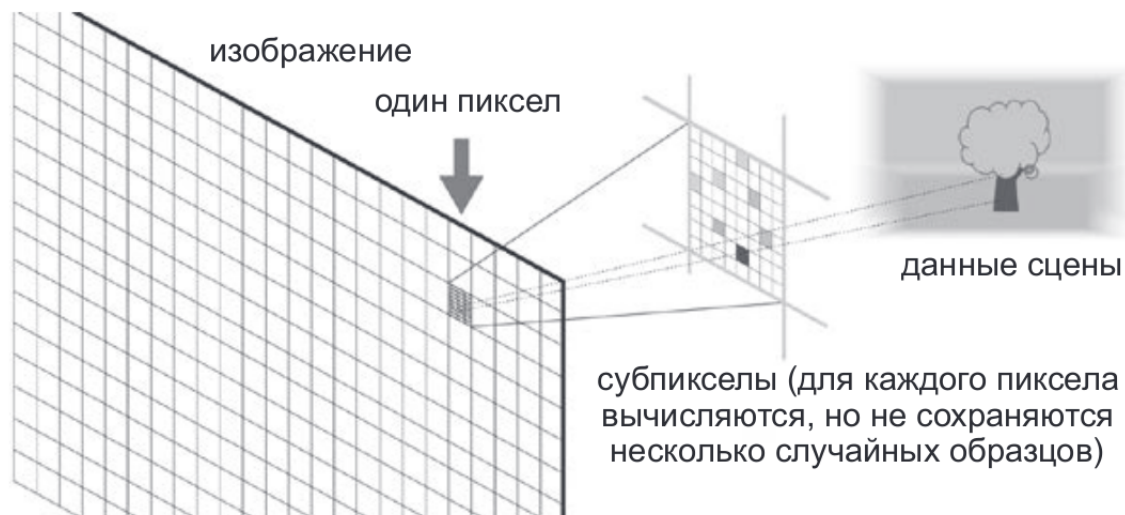


Рис. 1.1: Представление построения изображения методом трассировки лучей.

Хотя при отслеживании всего одного луча на пиксель будет построено правильное изображение, удаленные подробности (такие как дерево на рисунке 1.1) сцены будут потеряны. Кроме того, при перемещении камеры

объекта могут появляться шумовые эффекты [3]. На рисунке 1.2 представлено изображение с шумовыми эффектами.

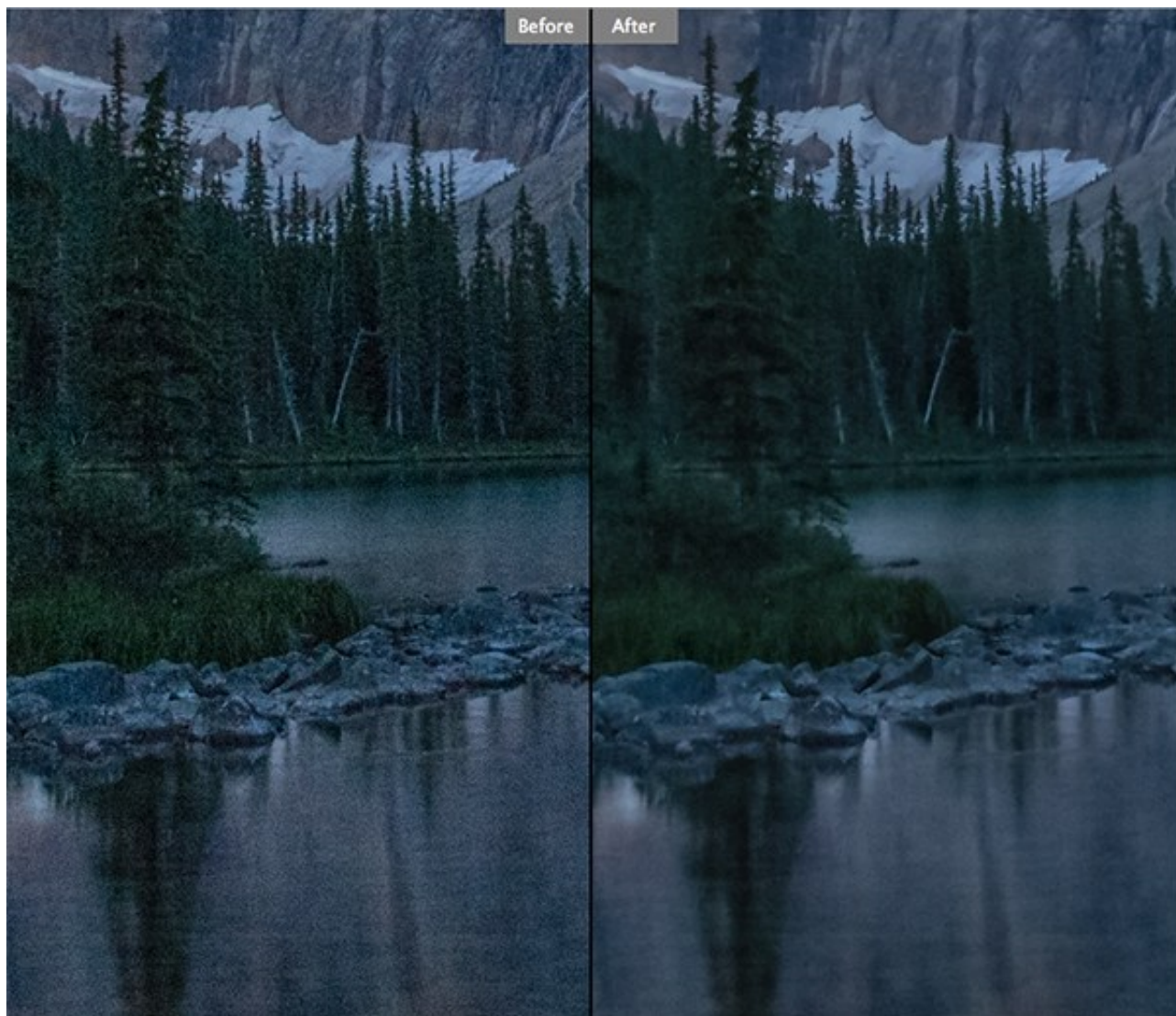


Рис. 1.2: Слева – изображение с шумовыми эффектами, справа – тоже самое изображение без шумовых эффектов.

### 1.1.2 Трассировка лучей методом Монте-Карло

Для того чтобы решить задачу без увеличения размеров изображения, программы трассировки лучей проецируют несколько лучей на пиксель, с небольшим изменением направлением для каждого луча. В таком случае, сохраняется только среднее значение цвета, а остальные значения не используются. На рисунке 1.1 схематично изображена проекция нескольких лучей на один пиксель.

Процес получения одного среднего значения называют избыточной выборкой (англ. supersampling), или выборкой по методу Монте-Карло. Трассировку лучей с использованием такого алгоритма называют трассировкой лучей методом Монте-Карло. Чем больше образцов (англ. sample) будет получено, тем ниже будет уровень шума в итоговом изображении. Избыточная выборка – важный шаг в процессе обработки изображений, синтезированных компьютером.

В избыточной выборке возможна параллельная обработка – подсчет результатов проекции множества лучей на сцену, данную задачу очень легко распараллелить. Но, в конечном итоге используются не отдельные результаты, а только их среднее значение.

Избыточная выборка является задачей где можно эффективно применить квантовые вычисления [4]. Это становится возможным, потому что квантовые вычисления обладают свойством квантовой суперпозиции [5] и принципом квантовой запутанности [6].

## **1.2 Квантовая обработка изображений**

Технология квантовой обработки изображений использует квантовые вычисления для расширения возможностей обработки и синтеза изображений. Данная технология находится на начальной стадии развития [7], в связи с этим на нас наложен ряд ограничений. Во-первых, в настоящий момент, квантовые компьютеры не доступны для широкого использования. Из-за этого, все квантовые вычисления мы можем только лишь эмулировать. Во-вторых, для эмуляции квантовых вычислений нужно большое количество оперативной памяти, размер которой растет экспоненциально [8]. В связи с этим мы ограничены вычислительными мощностями современных компьютеров.

### **1.2.1 Использование избыточной выборки**

Одним из применений квантовых вычислений, которое подходит под условие поставленной задачи, является квантовая избыточная выборка (ан-

гл. quantum supersampling). Метод квантовой избыточной выборки в итоге получает результат примерно такой же, как и у существующего классического аналога [4]. Но, изображение полученное квантовой выборкой обладает другими преимуществами: средний уровень шума в изображениях, полученных в результате традиционной выборки примерно одинаков, но характер шума сильно различается [4]. В изображениях с квантовой выборкой некоторые пиксели сильно зашумлены, тогда как цвет остальных определен без каких-либо ошибок вообще.

Таким образом, квантовая выборка выигрывает у традиционной при необходимости пост-обработки синтезируемого изображения, например, удаления всего видимого шума. В случае квантовой выборки этот процесс относительно прост, так как на полученном изображении будут пиксели изображенные без каких-либо ошибок либо сильно выделяющиеся пиксели. Сравнение выборок приведено на рисунке 1.3.

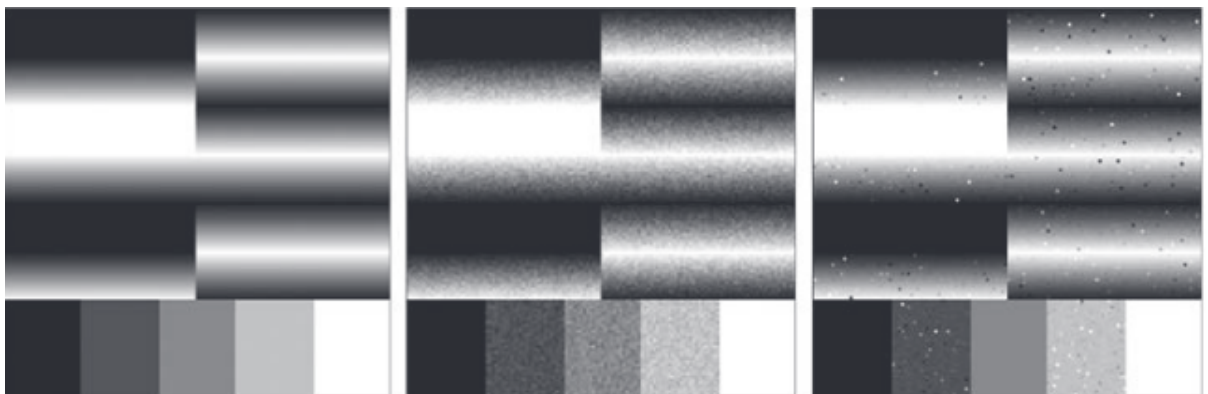


Рис. 1.3: Изменение характера шума. Первое изображение – эталонное; второе – выборка по методу Монте-Карло; третье – квантовая выборка.

К сожалению, для реализации полноценного квантового трассировщика лучей, потребуется намного больше квантовых мощностей, чем доступно в данный момент [9]. Из-за этого, в данной работе возможно реализовать только метод квантовой избыточной выборки в качестве той части алгоритма, которую возможно заменить на квантовую.



## 1.3 Основные положения квантовых вычислений

### 1.3.1 Квантовый бит

В квантовых вычислениях физические свойства квантовых объектов реализованы в кубитах (англ. quantum bit). Классический бит принимает только два значения – 0 или 1. Кубит до измерения принимает одновременно оба значения. Из-за этого, кубит принято обозначать выражением  $\alpha|0\rangle + \beta|1\rangle$ , где  $\alpha$  и  $\beta$  — комплексные числа, удовлетворяющие условию (1.1)

$$|\alpha|^2 + |\beta|^2 = 1. \quad (1.1)$$

Измерение кубита мгновенно «схлопывает» его состояние в базисное – 0 или 1. Вероятности перехода в эти состояния равны соответственно  $|\alpha|^2$  и  $|\beta|^2$ .

На рисунке 1.4 представлена возможная интерпретация состояния кубита в графической форме. Возможные состояния кубита представлены в виде кругов, серое наполнение – вероятность нахождения кубита в данном состоянии.

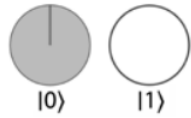
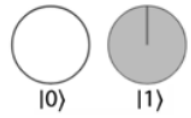
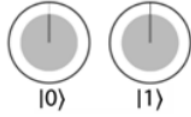
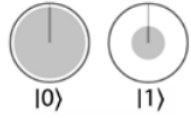
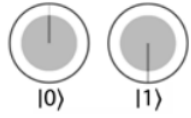
Возможные значения кубита	Графическое представление
$ 0\rangle$	
$ 1\rangle$	
$0.707 0\rangle + 0.707 1\rangle$	
$0.95 0\rangle + 0.35 1\rangle$	
$0.707 0\rangle - 0.707 1\rangle$	

Рис. 1.4: Возможные значения кубита и их представление.

### 1.3.2 Квантовый регистр

На кубиты может быть наложена ненаблюдаемая связь – при всяком изменении над одним из нескольких кубитов остальные меняются согласованно с ним. Таким образом, можно интерпретировать такую совокупность кубитов как квантовый регистр. Такой регистр может находиться во всех комбинациях составляющих его битов, и, кроме этого, реализовывать зависимости между ними.

На рис 1.5 показано одно из возможных представлений квантовых регистров. Это представление использует введенное нами представление кубитов в виде кругов из рисунка 1.4.

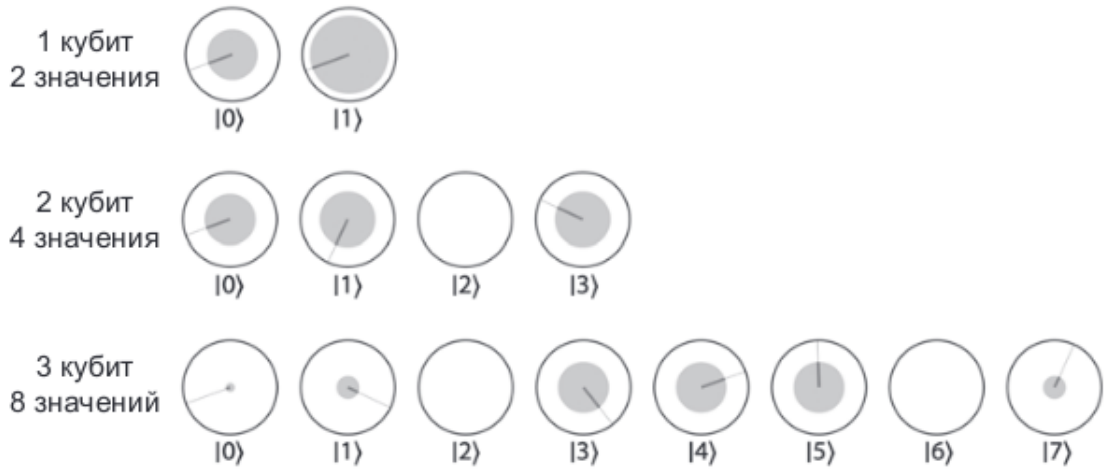


Рис. 1.5: Графическое представление квантового регистра.

Согласно принципу суперпозиции квантовый регистр из  $n$  кубитов может находиться и во многих других состояниях, которые описываются волновыми функциями, являющимися линейными комбинациями базисных волновых функций (1.2):

$$\Psi = \sum_{\vec{i}=|00\dots0\rangle}^{\vec{i}=|11\dots1\rangle} A \rightarrow_i \Psi(|\vec{i}\rangle) \quad (1.2)$$

где:

- $\vec{i}$  –  $n$ -разрядные двоичные коды, которые пробегают все возможные значения от  $|00\dots0\rangle$  до  $|11\dots1\rangle$ ;
- $A \rightarrow_i$  – комплексные амплитуды, которые должны удовлетворять условию (1.3).

$$\sum_{\vec{i}=|00\dots0\rangle}^{\vec{i}=|11\dots1\rangle} |A \rightarrow_i|^2 = 1 \quad (1.3)$$

## 1.4 Синтез изображения в квантовом представлении

### 1.4.1 Квантовый пиксельный шейдер

Пиксельный шейдер (англ. pixel shader) – это программа (которая чаще всего выполняется на графическом процессоре), которая на вход принимает координаты  $x$  и  $y$  и на выходе выдает цвет пикселя, находящегося в заданных координатах. Для реализации квантового пиксельного шейдера, воспользуемся двумя квантовыми регистрами, каждый из которых состоит из  $N$  кубитов. Назовём их  $qx$  и  $qy$  для осей  $x$  и  $y$  соответственно. Таким образом, размер синтезируемого изображения зависит от величины  $N$ . Например, при  $N = 4$ , регистры  $qx$  и  $qy$  будут содержать  $2^N = 2^4 = 16$  кубитов, что будет соответствовать размеру изображения 16x16 пикселей. Такой пиксель может быть только черным (1) или белым (0), потому что он будет соответствовать значениям которые может принимать один кубит. На рисунке 1.6 представлен пустой холст в квантовом представлении.

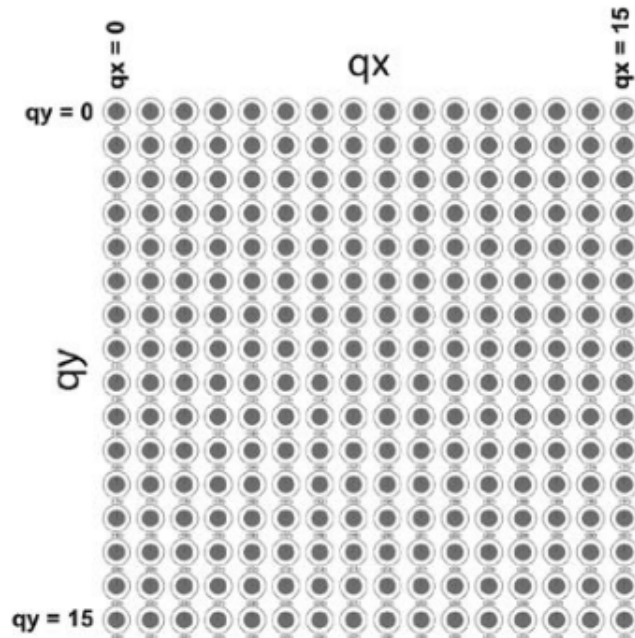


Рис. 1.6: Изображение холста в квантовом представлении.

## 1.4.2 Квантовая фаза

Произвольное квантовое состояние, обозначаемое  $|\psi\rangle$ , может быть любая суперпозиция, записанная в виде (1.4):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (1.4)$$

базисных векторов. Согласно условию (1.1) и основываясь на том факте, что глобальная фаза не наблюдаема (то есть  $|\psi\rangle$  тоже самое что и  $e^{j\gamma}|\psi\rangle$ ) [10], выражение (1.4) можно переписать в виде (1.5):

$$|\psi\rangle = \sqrt{1-p}|0\rangle + e^{j\gamma}\sqrt{p}|1\rangle \quad (1.5)$$

где  $0 \leq p \leq 1$  – вероятность того, что бит находится в состоянии 1 и  $0 \leq \psi < 2\pi$  – квантовая фаза (англ. quantum phase).

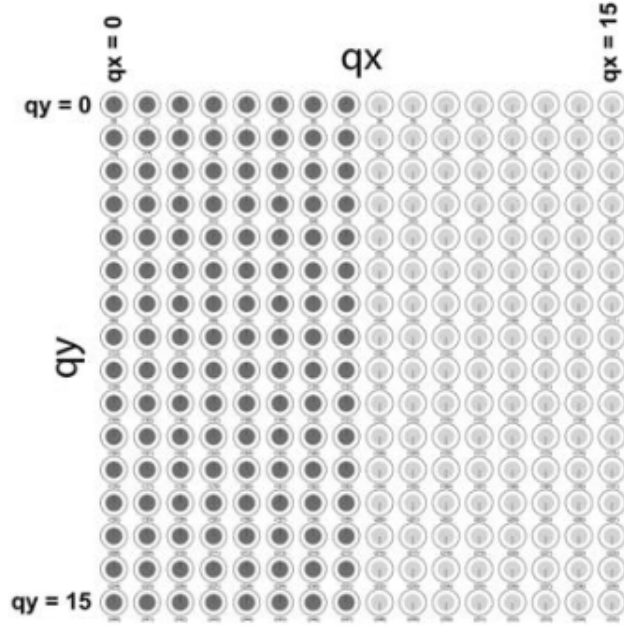


Рис. 1.7: Переключение фазы для половины изображения.

Применив операцию смены фазы на противоположную (то есть повернуть кубит на 180 градусов), можно изменить состояние кубита аналогично на противоположное. Операцию смены фазы можно применять сразу на несколько кубитов. Таким образом, всего за одну операцию, можно, например, изменить цвет половины (или сразу всех) пикселей синтезируемого

изображения. Выигрыш во времени по сравнению с обычными вычислениями очевиден. Используя квантовую фазу можно изображать прямые в квантовом представлении. Пример использования квантовой фазы представлен на рисунке 1.7.

### 1.4.3 Усиление комплексной амплитуды

Допустим, что у нас имеется четырехкубитный квантовый регистр, который содержит одно из трех квантовых состояний, но мы не знаем какое именно. В каждом из этих состояний присутствует некоторое значение с инвертированной фазой. Назовем его помеченным значением (рисунок 1.8). При чтении из квантового регистра, мы получим случайно число с равномерным распределением, и ничего не сможем узнать о том, какое из трех квантовых состояний было исходным.

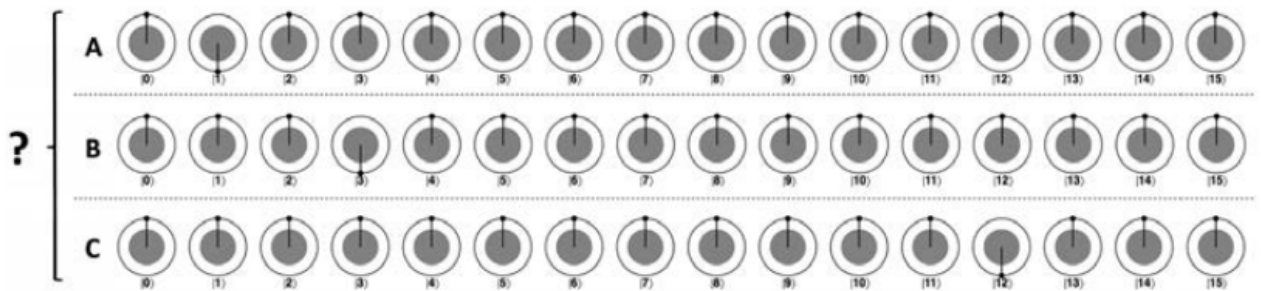


Рис. 1.8: Четырехкубитный регистр в трех состояниях, каждое из которых содержит помеченное значение.

Введем зеркальную операцию. Это операция выполняет следующие действия: берёт регистр, находящийся в состоянии  $|0\rangle$  и помечает одно из значений регистра с помощью операции смены фазы на противоположную (рисунок 1.9). Теперь амплитуды в каждом состоянии очень сильно различаются, благодаря чему выполнение операции чтения из регистра с большой вероятностью покажет, у какого значения инвертирована фаза – а следовательно, в каком из состояний находился регистр изначально.

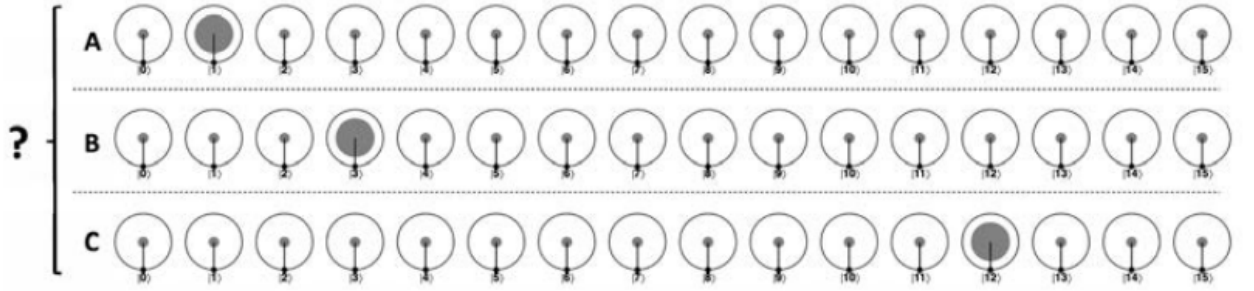


Рис. 1.9: Разность амплитуд после применения зеркальной операции разности фаз.

Применении зеркальной операции повторно вернет кубит в исходное состояние. Но, допустим, что при перед повторным использованием так же будет повторна применена операция смены фазы. В результате получится еще одна разность фазы, а вероятность успешного обнаружения помеченного значения возрастает [10]. На рисунке 1.10 показано, что получается при двукратном применении комбинации инвертирования и зеркальной операции.

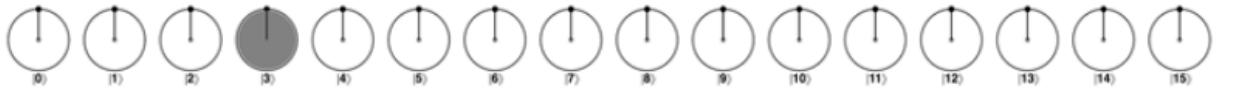


Рис. 1.10: Повторное применение комбинации инвертирования и зеркальной операции ко второму состоянию.

Две эти операции образуют мощную комбинацию. Применяя их вместе, можно увеличивать шанс правильного чтения помеченного значения. Но, к сожалению, бесконечно увеличивая число итераций комбинаций этих операций, невозможно достичь 100% вероятности правильного чтения значения. В начале итераций, разность амплитуд увеличиваются, и благодаря этому растет вероятность обнаружения правильного значения. Начиная с итерации  $N_{AA}$  разность амплитуд уменьшается, и с каждой итерацией вероятность все ближе становится к исходной, а потом вовсе становится исходной (той, что была без использования итераций) [11].

Число  $N_{AA}$  можно обозначить как оптимальное количество итераций при усилении комплексной амплитуды. Оно выражается с помощью фор-

мулы (1.6):

$$N_{AA} = \frac{\pi\sqrt{2^n}}{4} \quad (1.6)$$

где  $n$  – количество кубитов.

#### 1.4.4 Квантовая фазовая логика

Квантовая фазовая логика инвертирует фазу каждого входного значения, которое дает 1 в результате. Фазовая логика принципиально отличается от любой традиционной логики – результаты логических операций скрыты в фазах и их невозможно прочесть. Но, при этом, инвертируя фазы в суперпозиции, можно пометить несколько решений в одном регистре. Кроме того, при использовании инвертирования и усиления комплексной амплитуды можно создавать результаты, доступные для чтения.

С помощью комбинации усиления амплитуды и операций фазовой логики, можно сохранить значение логической операции в фазе состояния [12]. Таким образом, мы можем описывать более сложные фигуры, например кривые.

Окружность задается уравнением вида (1.7):

$$x^2 + y^2 = r^2 \quad (1.7)$$

Предположим, что мы хотим заполнить все пиксели, находящиеся внутри окружности, то есть пиксели, подходящие под условие (1.8):

$$x^2 + y^2 < r^2 \quad (1.8)$$

Для выполнения этого действия нам потребуются выше описанные регистры  $qx$  и  $qy$ , а так же дополнительные регистр-аккумулятор  $qacc$ . Дальнейший алгоритм таков:

- инициализировать регистры  $qx$ ,  $qy$  и  $qacc$ ;
- ввести регистры  $qx$  и  $qy$  в суперпозицию;
- добавить в регистр  $qacc$  сумму квадратов регистров  $qx$  и  $qy$ ;



- вычесть из регистра  $qacc$  квадрат радиуса описываемой окружности;
- инвертировать регистр  $qacc$  для всех значащих битов;
- восстановить регистр  $qacc$ .

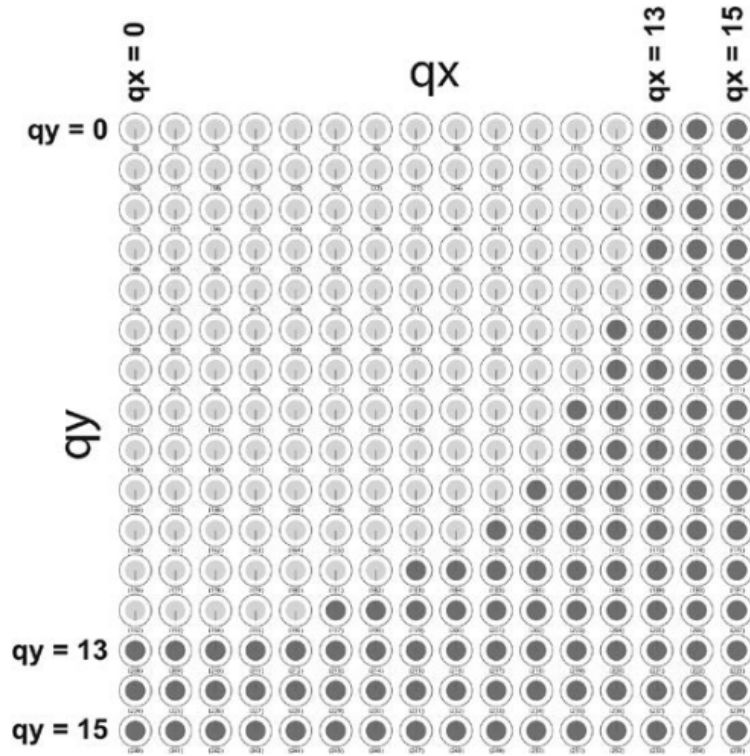


Рис. 1.11: Переключение кривых на холсте с помощью фазовой логики.

### 1.4.5 Квантовое преобразование Фурье

Квантовое преобразование Фурье позволяет обращаться к скрытой информации, хранящейся в фазах и амплитудах квантового регистра. Данный примитив предоставляет собственный механизм манипуляций с фазами кубитов [13]).

Допустим, имеется четырехкубитный квантовый регистр, содержащий одно из трех состояний, но точно неизвестно какое (рисунок 1.12). Как уже говорилось ранее, при чтении регистра будет получено случайное значение с равномерным распределением.

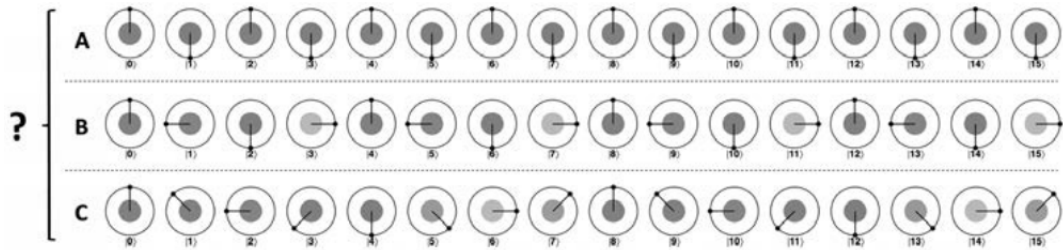


Рис. 1.12: Три разных состояния кубита до применения квантового преобразования Фурье

Усиление квантовой амплитуды в данном примере не принесет никакой пользы, так как нет какой-то одной фазы, которая бы выделялась на фоне других в каждом состоянии. Применение квантового преобразования Фурье к рассматриваемому регистру перед чтением результата преобразует каждое из состояний к результату, показанному на рисунке 1.13. Таким образом, можно однозначно определить, какое состояние было исходным.

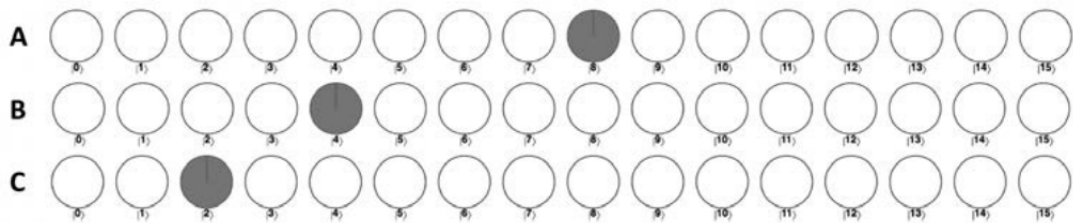


Рис. 1.13: Три разных состояния кубита после применения квантового преобразования Фурье

Между результатами на рисунке 1.12 можно заметить связь: в первом состоянии фаза входного состояния возвращается в 0 восемь раз, а квантовое преобразование Фурье позволяет прочесть значение 8. Во втором состоянии фаза возвращается к своему начальному состоянию четыре раза, а квантовое преобразование Фурье позволяет прочесть значение 4. Третье состояние подчиняется той же самой закономерности. Квантовое преобразование Фурье успешно открывает частоту сигнала [14], содержащуюся в квантовом регистре. Само по себе квантовое преобразование Фурье имеет сильное сходство с классическим механизмом обработки сигналов, называемым дискретным преобразованием Фурье [15].

## 1.5 Квантовая избыточная выборка

Избыточная выборка – процесс увеличения число дискретных выборок на пиксель. На один пиксель проецируется не один, а несколько лучей. Результат проекции каждого такого луча сохраняется в соответствующий субпиксель (англ. subpixel). После того, как все нужные выборки сохранены в экранном буфере, итоговый цвет пикселя определяется как усреднённый цвет всех соответствующих ему субпикселей. Таким образом, формула принимает вид (1.9):

$$res = \frac{sample_0 + sample_1 + \dots + sample_{n-1}}{n} = \frac{\sum_{i=0}^{n-1} sample_i}{n} \quad (1.9)$$

где:

- $res$  – итоговый цвет пикселя;
- $n$  – количество выборок на пиксель;
- $sample_i$  – цвет  $i$ -ой выборки.

В случае квантовой избыточной выборки, для каждого блока необходимо оценить количество субпикселей с инвертированной фазой. Для черных и белых субпикселей (представленных инвертированной или не инвертированной фазой) это позволит нам получить значение для каждого результирующего пикселя, характеризующее интенсивность исходных составляющих субпикселей.

Преимущества использования квантовой избыточной выборки (по сравнению с обычной) связано не с количеством операций графического вывода, а с различиями в характере наблюдаемого шума. В среднем, при сравнении двух идентичных синтезируемых изображения, погрешность на пиксель у квантовой избыточной выборки на 33% ниже, чем у метода Монте-Карло (обычная избыточная выборка) [16]. Помимо этого, количество пикселей с нулевой погрешностью в среднем в два раза больше, чем у метода Монте-Карло [16].

### 1.5.1 Принцип работы

Основополагающая идея квантовой избыточной выборки заключается в применении метода объединения итераций усиления комплексной амплитуды (1.4.3) с квантовым преобразованием Фурье (1.4.5). Квантовое преобразование Фурье позволит оценить количество элементов, инвертированных квантовой логикой, используемой в подсхеме инвертирования каждой итерации усиления комплексной амплитуды. В данном случае подсхемой инвертирования является программа, которая инвертирует фазу белых субпикселей.

Определим квантовый регистр, который будет выполнять роль «счётчика». Значение этого регистра будет определять, сколько итераций выполнит наша схема. Введя регистр в суперпозицию, будет выполнено суперпозиция разного количества итераций усиления комплексной амплитуды. Как уже было написано выше, вероятность чтения нескольких инвертированных значений в регистре зависит от количества выполняемых итераций. Кроме этого, колебания вводятся в зависимости от количества инвертированных значений. Таким образом, при выполнении суперпозиции разного количества итераций усиления комплексной амплитуды, вводятся периодические колебания по комплексным амплитудам квантового регистра с частотой, зависящей от количества инвертированных значений.

Для чтения частот, закодированных в квантовых регистрах, можно использовать квантовое преобразование Фурье [17]. Зная количество субпикселей, использованных в квантовой выборке, можно определить яркость анализируемого пикселя.

Качество выборки напрямую зависит от количества кубитов для «счётчиков». С увеличением количества образцов (кубитов) вероятность получения точного ответа растёт [18].

### 1.5.2 Поисковая таблица

При запуске квантового алгоритма избыточной выборки и в конце его выполнения читая значение квантового регистра, мы получаем число – оно

связано с количеством белых субпикселей в заданном блоке, но не будет точно равно ему.

Поисковая таблица квантовой выборки (англ. quantum supersampling lookup table) – инструмент для определения количества субпикселей в блоке, подразумеваемого считанным значением из квантового регистра. Например, поисковая таблица, для квантовой выборки с размером квантового шейдера  $4 \times 4$  и регистром счетчиком, состоящим из четырех кубитов, будет выглядеть как таблица с  $2^4 = 16$  столбцами и  $2^4 = 16$  строками. В строках поисковой таблицы перечисляются возможные результаты чтения значения из квантового регистра. В столбцах перечисляются возможные количества субпикселей в квантовом шейдере, которые могут привести к такому значению, который был получен путём чтения квантового регистра.

При получении значения из квантового регистра, в поисковой таблице выбирается строка соответствующая считанному значению. Далее, оценивая количество «белых» субпикселей, расположенные в этой строке (а точнее лишь вероятность нахождения этих субпикселей в анализируемом пикселе), выбирается конечная яркость пикселя. Из-за того что в строках расположены лишь вероятности, появляется некоторая погрешность и не всегда можно однозначно определить яркость пикселя.

Таким образом, можно описать функцию, на вход принимающую номер строки ( $k$ ) таблицы, и возвращающую вероятность яркости пикселя (1.10):

$$\gamma(k) = \sum_{i=0}^{n-1} x_{k_i} \quad (1.10)$$

Поисковая таблица – характерный признак алгоритма квантовой избыточной выборки. С увеличением размера квантового шейдера (следовательно увеличения количества субпикселей), растёт вероятность точного определения яркости выбранного пикселя.

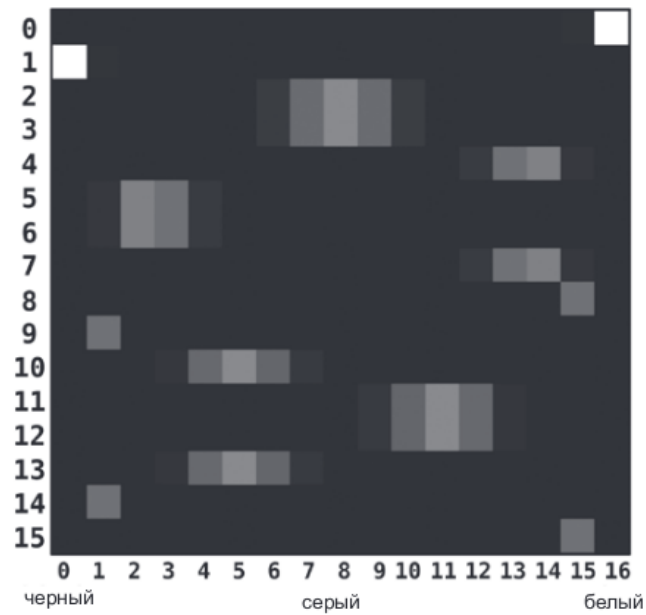


Рис. 1.14: Поисковая таблица квантовой выборки. По оси ОУ – результат квантовой выборки, по оси ОХ – цвет пикселя (сумма белых пикселей)

### 1.5.3 Карта достоверности

Поисковая таблица также может использоваться для получения вероятности того, что итоговая яркости пикселя выбрана правильно. По расположению считанного значения из квантового регистра, в строке таблицы можно оценить вероятность того, что полученное значение было правильным. Для каждого значения из выбранной строки имеется вероятность что данный субпиксель является белым – это можно сделать с помощью расположения считанного значения (в процессе избыточной выборки) в соответствующей строке таблицы поиска. По этим результатам можно построить «карту достоверности» (англ. confidence map), обозначающую вероятное расположение ошибок в синтезируемом изображении.

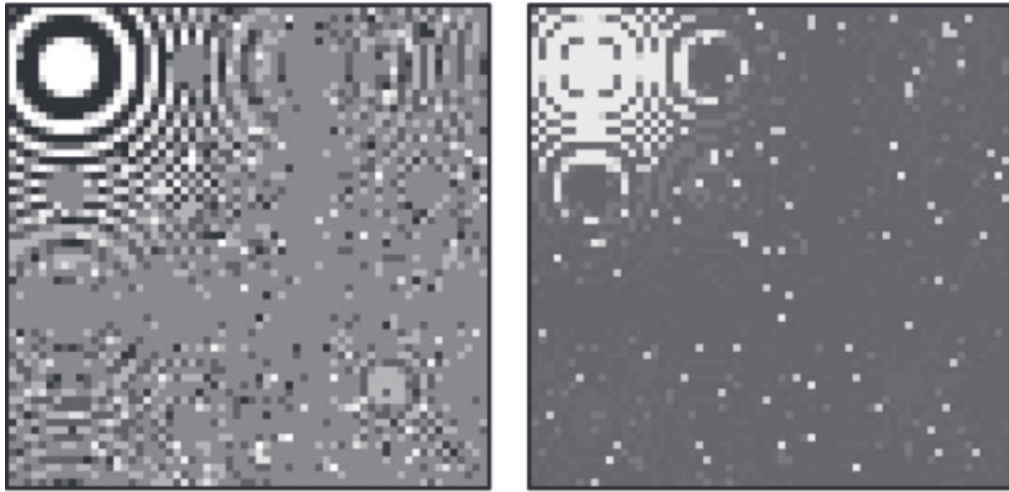


Рис. 1.15: Карта достоверности. Слева – результат квантовой выборки, справа – карта достоверности на уровне пикселей.

## 1.6 Колоризация изображения

Фазы и комплексные амплитуды квантового регистра можно использовать для кодирования более широкого диапазона цветовых значений (помимо белого и черного), но тогда метод квантовой избыточной выборки работать не будет [19].

Для колоризации изображения можно воспользоваться технологией битовых слоёв. Квантовый пиксельный шейдер будет использоваться для построения отдельных монохромных изображений, каждое из которых будет представлять один бит изображения. Таким образом, пиксельный шейдер фактически будет генерировать  $N$  монохромных изображений, где  $N$  – количество цветов, которые нужно «запутать» в изображении. Все эти  $N$  изображений пройдут квантовую избыточную выборку по отдельности и будут объединены в итоговое цветное изображение.

## Вывод

В данном разделе был проведен анализ квантовых алгоритмов и структур данных, которые возможно использовать в поставленной задаче. В качестве ключевого алгоритма, который может улучшить качество синтези-

руемого изображения, выбран алгоритм квантовой избыточной выборки, в сочетании с такими структурами данных как квантовая поисковая таблица и квантовая карта достоверности. Именно этот алгоритм и будет реализован в рамках данной работы.



## 2 Конструкторская часть

### 2.1 Схема квантового компьютера

Опыт, накопленный в квантовой физике 20 века, дает надежду на то, что «железо» (англ. hardware) для квантового компьютера возможно реализовать в недалеком будущем [20]. Однако, любой компьютер состоит не только лишь из одного так называемого «железа», и квантовый компьютер не является исключением. Если задача создания операционной системы для классических вычислений является такой же масштабной задачей, как создание «железа», то в квантовом случае операционная система представляет собой гораздо большую трудность [20].

Схематически квантовый компьютер можно представить в виде структуры, состоящей из трёх блоков:

- квантовый процессор;
- квантовая операционная система;
- пользовательский интерфейс.

#### 2.1.1 Квантовая коррекция ошибок

Как уже говорилось в аналитической части, квантовые вычисления выдают только лишь какую-то вероятность значения, а не само значение (это явление называется квантовой декогеренцией [21]). В связи с этим была придумана квантовая коррекция ошибок [22], которая используется для защиты от ошибок из-за квантовой декогеренции.

#### 2.1.2 Пользовательский интерфейс квантового компьютера

Р. Фейман в своей работе [23] предложил пользовательский интерфейс, основанный на массиве квантовых гейтов, которые реализуют простейшие

унитарные операторы на малом числе кубитов. Такой интерфейс следует из современного понимания квантовой теории [20] и в настоящее время является основным. Важным свойством этого интерфейса является тот факт, что он задействует малое количество кубитов.

К сожалению, такой интерфейс подходит для систем с декогерентностью. С декогерентностью частично можно бороться [22] – для этого существуют квантовая коррекция ошибок. Но использование квантовой коррекции ошибок требует дополнительных кубитов, что является проблематичным требованием. Кроме того, к декогерентности оказываются особо чувствительны именно те квантовые состояния, которые возникают при реализации квантовых вычислений [20].

### **2.1.3 Квантовая операционная система**

Квантовая операционная система должна работать в режиме реального времени [20]. Это предполагает использование специальных программных примитивов, вместо массива гейтов. Как уже было указано в главе 2.1.2, основной квантовой интерфейс на данный момент основан на использовании массива квантовых гейтов, что приводит нас к противоречию.

Ядро квантовой операционной системы является программа, написанная для классического суперкомпьютера. Программные примитивы с высокой точностью моделируют динамику малого фрагмента всей модели, а ядро операционной системы управляет динамикой квантового состояния всей системы, причем на большом отрезке времени. На этом моменте мы сталкиваемся со второй проблемой квантовой операционной системы: на данный момент неизвестно, как квантовая теория работает в области сложных (в том числе длительных) процессов. Существует лишь математическая схема квантовых алгоритмов [20].

Таким образом, квантовый компьютер становится лишь экспериментальным устройством, на создание которого уйдет ещё какое-то время. В настоящее время мы можем лишь эмулировать квантовые вычисления с использованием обычных компьютеров.

## 2.2 Эмуляция квантовых вычислений

Квантовые вычисления возможно эмулировать на обычном компьютере [24]:

- состояние кубита можно представить комплексным числом, занимающим от  $2N$  бит, где  $N$  – разрядность процессора;
- состояние из  $K$  связанных кубитов можно представить в виде  $2^K$  комплексных чисел;
- квантовую операцию над  $M$  кубитами можно представить матрицей  $2^M \times 2^M$ .

В таком случае, для хранения эмулируемых состояний:

- 10 кубитов нужны 8 Кб;
- 20 кубитов нужны 8 Мб;
- 30 кубитов нужны 8 Гб;
- и так далее.

Таким образом, предел симуляции квантового компьютера на классических компьютерах обусловлен количеством оперативной памяти [25]. Можно сделать вывод, что при эмуляции квантовых вычислений в выбранном мною алгоритме можно будет воспользоваться лишь не более чем 30 – 32 кубитами.

Существует много различных реализаций квантовых вычислений [25], но все они основаны на одних и тех же принципах:

- используется некоторая схема хранения всех состояний кубитов;
- все операции над кубитами обеспечиваются с помощью специальных унитарных матриц [26];
- используется рандомизатор для внесения неопределенности в квантовой системе;

### **2.2.1 Квантовое превосходство**

Квантовое превосходство – способность квантовых вычислительных устройств решать проблемы, которые классические компьютеры практически не могут решить [27]. Достижение квантового превосходства означает, что задачи, такие как, например, факторизацию больших чисел можно решать за адекватное время. Кроме этого, квантовый компьютер с запущенной на нем некоторой квантовой схеме, то результат этой работы будет невозможно смулировать на обычном компьютере, то есть классический компьютер воссоздать результат работы такой схемы будет не в состоянии.

## **2.3 Структуры данных для квантового алгоритма избыточной выборки**

### **2.3.1 Квантовая поисковая таблица**

Квантовую поисковую таблицу можно реализовать в виде обычной матрицы. Такая реализация проста и вытекает из главы 1.5.2. В строках матрицы будут храниться результаты чтения значения из квантового регистра, а в столбцах перечислены возможные количества субпикселей в квантовом шейдере, которые могут привести к такому значению, который был получен путём чтения квантового регистра.

Во время синтеза изображения, в строках матрицы будет постоянно проводиться поиск нужного значения. Особенно, если данная квантовая таблица будет использоваться несколько. Для ускорения поиска, после заполнения строк матрицы, можно отсортировать их и в дальнейшем пользоваться бинарным поиском [28], что ускорит итоговое время работы алгоритма.

### 2.3.2 Квантовая карта достоверности

Квантовую карту достоверности реализуют в виде матрицы, так как очень легко провести параллель этой матрицы на синтезируемое изображение. Для каждой ячейки матрицы будет описывать пиксель в итоговом изображении. Номер строки соответствует координате  $x$  в итоговом изображении, номер столбца, соответственно, координату  $y$ . Сама ячейка матрицы хранит вероятность, что цвет данного пикселя был определен правильно.

## 2.4 Алгоритм квантовой избыточной выборки

В общем виде псевдокод алгоритма квантовой избыточной выборки (1) можно описать следующим образом:

---

**Алгоритм 1** Квантовая избыточная выборка

---

```
1: for all цвета колоризации do
2:   for all пиксели холста do
3:     перевести пиксель в состояние 1
4:   end for
5:   инициализировать регистр, хранящий количество итераций усиления квантовой амплитуды
6:   инициализировать поисковую таблицу и карту достоверности
7:   сформировать и заполнить квантовую поисковую таблицу
8:   for all пиксели холста do
9:     посчитать количество инвертируемых пикселей
10:    использовать эту величину для определения яркости пикселя
11:    заполнить соответствующую ячейку карты достоверности
12:   end for
13: end for
14: объединить полученные изображения в единое цветное изображение
```

---

## 2.4.1 Формирование квантовой поисковой таблицы

Ниже представлен алгоритм (2) формирования поисковой таблицы. Алгоритм следует оформить в виде подпрограммы.

---

**Алгоритм 2** Формирование квантовой поисковой таблицы

---

- 1: инициализировать и ввести в суперпозицию вспомогательные регистры  $qcount$  и  $qxy$
  - 2: выполнить усиление квантовой амплитуды на регистр  $qxy$  столько раз, из скольких битов состоит регистр  $qcount$
  - 3: применить квантовое преобразование Фурье на регистр  $qcount$
  - 4: прочитать значение каждого из битов  $qcount$  и заполнить ими ячейки таблицы
- 

## 2.4.2 Формирование квантовой карты достоверности

В алгоритме (3) представлен псевдокод формирования квантовой карты достоверности.

---

**Алгоритм 3** Формирование квантовой карты достоверности

---

- 1: считать все значения из таблицы поиска в заданной строке
  - 2: поделить сумму этих значений на длину строки
  - 3: на основе полученного посчитать вероятность ошибки в заданном пикселе и занести его в соответствующую ячейку карты достоверности
- 

## Вывод

В данном разделе было рассмотрено устройство квантового компьютера и был сделан вывод, что на данный момент это лишь гипотетическое устройство. Была рассмотрена схема эмуляции квантовых вычислений на обычных компьютерах и на основе этой информации описан алгоритм квантовой супер выборки с учетом выявленных и описанных ограничений эмуляции таких вычислений. Кроме того, были описаны структуры данных, которые понадобятся для реализации данного алгоритма.

## 3 Технологическая часть

В данном разделе представлены средства разработки программного обеспечения, детали реализации и тестирование функций.

### 3.1 Средства реализации

В качестве языка программирования, на котором будет реализовано программное обеспечение, выбран язык программирования JavaScript [29]. Выбор языка обусловлен тем, что для этого языка существует библиотека QCEngine [30], предоставляющая полный функционал который нужен для реализации выбранного мною алгоритма. В качестве программной платформы, с помощью которого можно превратить JavaScript из узкоспециализированного языка (работающего только в браузере) в язык общего назначения, был выбран Node.js [31].

Для создания пользовательского интерфейса программного обеспечения будет использоваться модуль node-gtk [32] вместе с модулем canvas [33]. Два этих инструмента в связке друг с другом позволят одновременно выводить синтезируемое изображение на экран, и сохранять его (при необходимости) в файл, например для дальнейшего анализа с помощью каких-либо других утилит.

Для тестирования программного обеспечения будет использоваться фреймворк Jest [34]. Данный инструмент предоставляет много различного функционала, позволяющего тестировать приложения, в том числе написанных на Node.js. Разрабатываемое программное обеспечение будет тестироваться по средствам модульного тестирования [35]. Функциональное тестирование [36] ПО проводиться не будет из-за своей специфики – разработанное ПО является GUI-приложением, что усложняет процесс тестирования.

Для обеспечения качества кода был использован инструмент ESLint [37], позволяющий во время процесса написания исходных кодов программного обеспечения контролировать наличие синтаксических и логических ошибок.

В качестве среды разработки выбран текстовый редактор Visual Studio Code [38], содержащий большим количеством плагинов и инструментов для

различных языков программирования, в том числе JavaScript. Такие инструменты облегчают и ускоряют процесс разработки программного обеспечения [39].

## 3.2 Детали реализации

В листингах 3.1 – 3.3 приведен исходный код реализации алгоритма квантовой супер выборки. Сам алгоритм разделен на подпрограммы: формирование квантовой поисковой таблицы, синтез изображения и заполнения карты достоверности.

```
1 function create_qss_lookup_table()
2 {
3     qc.reset((Constants.res_aa_bits + Constants.res_aa_bits) + Constants.
4         num_counter_bits);
5     var qxy = qc.new_qint(Constants.res_aa_bits + Constants.res_aa_bits, '
6         qxy');
7     var qcount = qc.new_qint(Constants.num_counter_bits, 'count');
8
9     var num_subpixels = 1 << (Constants.res_aa_bits + Constants.res_aa_bits)
10    qss_full_lookup_table = null;
11
12    for (var hits = 0; hits <= num_subpixels; ++hits)
13    {
14        create_table_column(hits, qxy, qcount);
15    }
16
17    var cw = qss_full_lookup_table;
18    qss_count_to_hits = [];
19
20    for (var count = 0; count < cw.length; ++count)
21    {
22        var best_hits = 0;
23        var best_prob = 0;
24
25        for (var hits = 0; hits < cw[0].length; ++hits)
26        {
27            if (best_prob < cw[count][hits])
28            {
29                best_prob = cw[count][hits];
30                best_hits = hits;
31            }
32        }
33    }
34 }
```



```

32     qss_count_to_hits.push(best_hits);
33 }
34
35 if (qss_full_lookup_table && images.display_cwtable)
36 {
37     images.display_cwtable.setup(cw[0].length, cw.length, 16);
38
39     for (var y = 0; y < cw.length; ++y)
40     {
41         for (var x = 0; x < cw[0].length; ++x)
42         {
43             images.display_cwtable.pixel(x, y, cw[y][x]);
44         }
45     }
46 }
47 }
48
49
50 function create_table_column(color, qxy, qcount)
51 {
52     var num_subpixels = 1 << (Constants.res_aa_bits + Constants.res_aa_bits)
53     var true_count = color;
54
55     qc.write(0);
56     qcount.hadamard();
57     qxy.hadamard();
58
59     for (var i = 0; i < Constants.num_counter_bits; ++i)
60     {
61         var reps = 1 << i;
62         var condition = qcount.bits(reps);
63         var mask_with_condition = qxy.bits().or(condition);
64
65         for (var j = 0; j < reps; ++j)
66         {
67             flip_n_terms(qxy, true_count, condition);
68             grover_iteration(qxy.bits(), mask_with_condition);
69         }
70     }
71
72     invQFT(qcount);
73     var table = [];
74
75     for (var i = 0; i < (1 << Constants.num_counter_bits); ++i)
76     {
77         table.push(qcount.peekProbability(i));
78     }
79

```

```

80  if (qss_full_lookup_table == null)
81  {
82      qss_full_lookup_table = [];
83
84      for (var i = 0; i < (1 << Constants.num_counter_bits); ++i)
85      {
86          qss_full_lookup_table.push([]);
87
88          for (var j = 0; j < num_subpixels; ++j)
89          {
90              qss_full_lookup_table[i].push(0);
91          }
92      }
93  }
94
95  for (var col = 0; col < 1 << Constants.num_counter_bits; ++col)
96  {
97      qss_full_lookup_table[col][true_count] = table[col];
98  }
99  }

```

Листинг 3.1: Функция формирования квантовой поисковой таблицы

```

1  function do_qss_image()
2  {
3      qc.reset(2 * Constants.res_aa_bits + Constants.num_counter_bits +
4          Constants.accum_bits);
5
6      var sp = {};
7
8      sp.qx = qc.new_qint(Constants.res_aa_bits, 'qx');
9      sp.qy = qc.new_qint(Constants.res_aa_bits, 'qy');
10     sp.counter = qc.new_qint(Constants.num_counter_bits, 'counter');
11     sp.qacc = qc.new_qint(Constants.accum_bits, 'scratch');
12     sp.qacc.write(0);
13
14     var total_pixel_error = 0;
15     var num_zero_error_pixels = 0;
16     qss_raw_result = [];
17
18     for (sp.ty = 0; sp.ty < Constants.res_tiles; ++sp.ty)
19     {
20         qss_raw_result.push([]);
21
22         Constants.update_fraction();
23
24         for (sp.tx = 0; sp.tx < Constants.res_tiles; ++sp.tx)
25         {
26             qss_tile(sp);
27             qss_raw_result[sp.ty].push(sp.readVal);
28         }
29     }
30 }

```

```

26     images.display_qss.pixel(sp.tx, sp.ty, sp.color);
27
28     if (ideal_result)
29     {
30         var pixel_error = Math.abs(sp.hits - ideal_result[sp.ty][sp.tx]);
31
32         if (pixel_error)
33         {
34             total_pixel_error += pixel_error;
35         }
36         else
37         {
38             num_zero_error_pixels++;
39         }
40     }
41 }
42 }
43 }

```

Листинг 3.2: Функция синтеза изображения на основе значений поисковой таблицы

```

1 function draw_confidence_map()
2 {
3     var cw = qss_full_lookup_table;
4
5     for (var ty = 0; ty < Constants.res_tiles; ++ty)
6     {
7         for (var tx = 0; tx < Constants.res_tiles; ++tx)
8         {
9             var qss_out = qss_raw_result[ty][tx];
10            var row_total = 0;
11            var row_max = 0;
12
13            for (var x = 0; x < cw[0].length; ++x)
14            {
15                row_total += cw[qss_out][x];
16
17                if (cw[qss_out][x] > row_max)
18                {
19                    row_max = cw[qss_out][x];
20                }
21            }
22
23            images.display_confidence.pixel(tx, ty, row_max / row_total);
24        }
25    }
26 }

```

## Вывод

В данном разделе были рассмотрены средства реализации программного обеспечения и листинги исходных кодов программного обеспечения, разработанного на основе алгоритма, изложенного в конструкторской части.

## 4 Исследовательская часть

В данном разделе будут приведены результаты работы разработанного программного обеспечения и поставлен эксперимент по сравнению уровня и характера шума синтезируемого изображения с использованием классического алгоритма (метода Монте - Карло) и квантового алгоритма избыточной выборки на различных размерах изображения.

### 4.1 Результаты работы программного обеспечения

На рисунке 4.1 приведено эталонное изображение, с которого будет проводиться квантовая избыточная выборка.

Рис. 4.1: Эталонное изображение размером 1024x512

На изображениях 4.2 - 4.6 приведен результат работы ПО для разных размеров изображений.

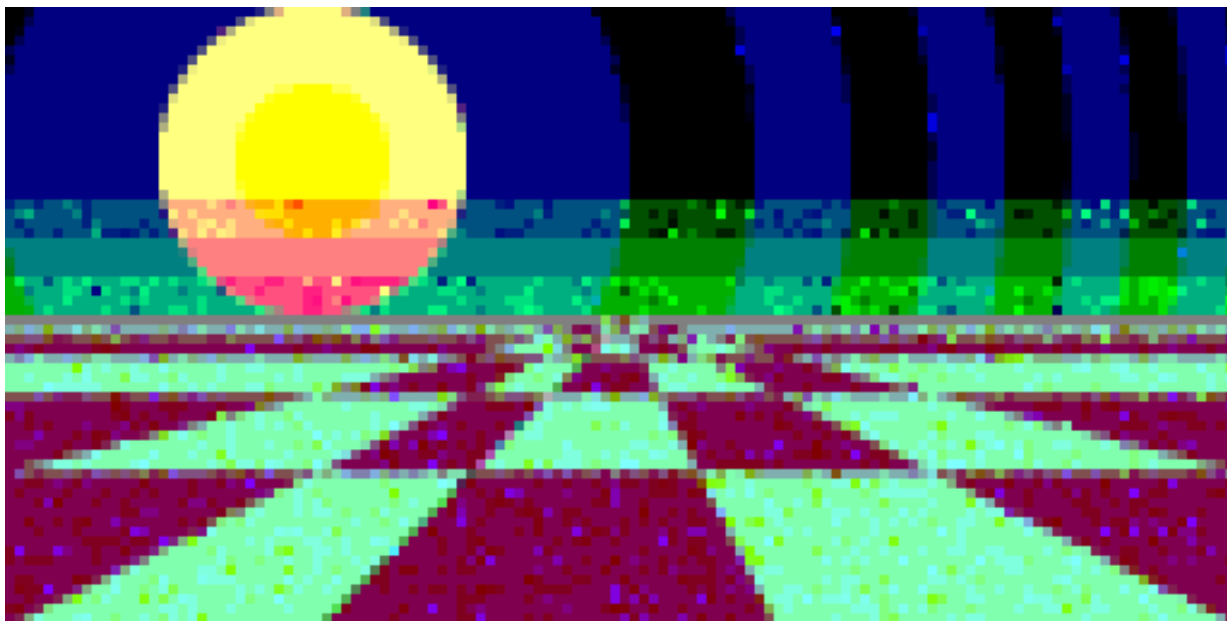


Рис. 4.2: Результат работы квантового алгоритма, 512x256, 4 итераций увеличения комплексной амплитуды

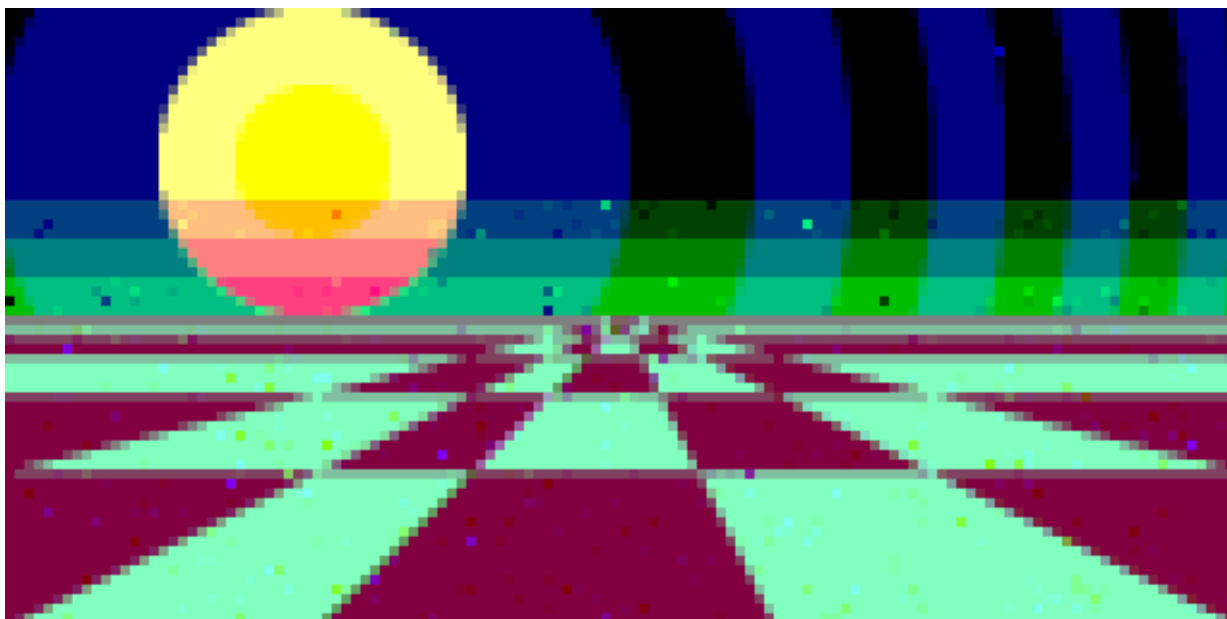


Рис. 4.3: Результат работы квантового алгоритма, 512x256, 6 итераций  
увеличения комплексной амплитуды

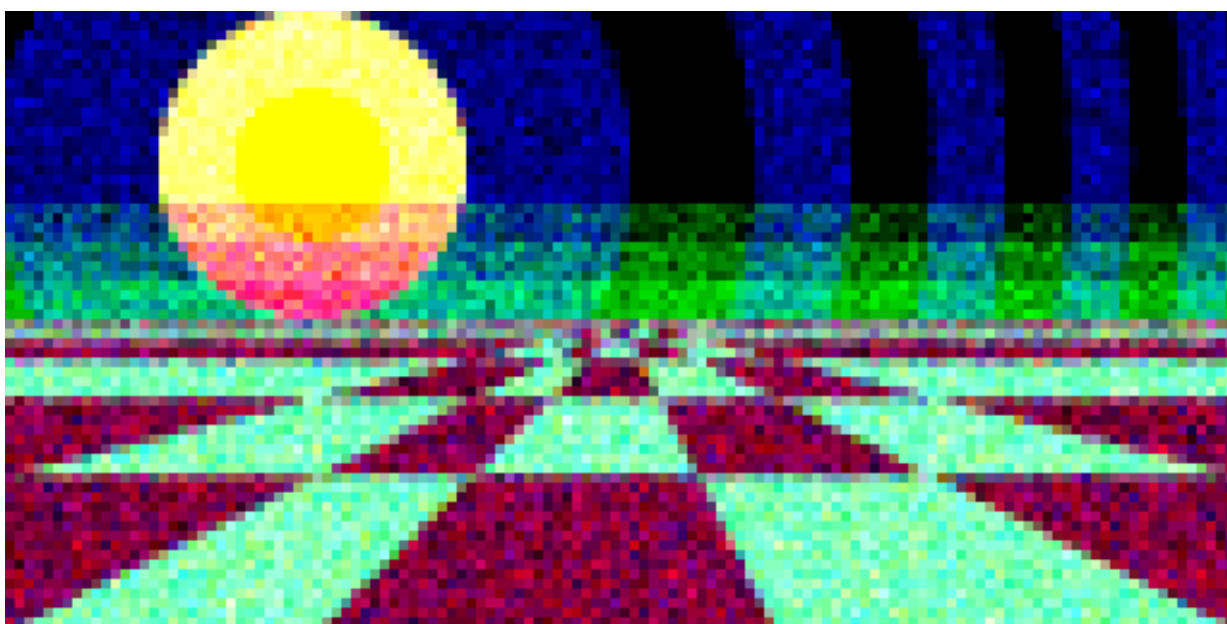


Рис. 4.4: Результат работы алгоритма Монте – Карло, 512x256

Рис. 4.5: Результат работы квантового алгоритма, 1024x512, 4 итерации  
увеличения комплексной амплитуды

Рис. 4.6: Результат работы алгоритма Монте – Карло, 1024x512

## 4.2 Постановка эксперимента

### 4.2.1 Цель эксперимента

Целью эксперимента является проведение трех независимых сравнений результатов работы программного обеспечения:

- сравнение уровня шума на изображении синтезируемого алгоритмом Монте – Карло и квантовым алгоритмом;
- сравнение характера шума на изображении синтезируемого алгоритмом Монте – Карло и квантовым алгоритмом;
- сравнить вычислительные сложности алгоритма Монте – Карло и квантового.

### 4.2.2 Сравнение уровня зашумленности

Сравнить уровень зашумленности двух синтезируемых изображений можно рассчитав средний процент ошибки на каждый пиксель.

Результаты сравнения уровня зашумленности для изображений разных размеров приведены в таблице 4.1. Во втором и третьем столбце таблицы расположены средние проценты ошибок для рассматриваемых реализаций алгоритмов. Размер сабпикселя равен 4x4.

### 4.2.3 Сравнение характера шума

Из-за того что в квантовом случае пиксели получаемого изображения могут быть либо идеальными, либо крайне зашумленными [4], сравнение

Таблица 4.1: Сравнение уровня зашумленности для классического и квантового алгоритма избыточной выборки.

Размер изображения	Классическая выборка	Квантовая выборка
64x64	3%	2%
128x128	4%	2%
256x256	4%	3%
512x512	4%	3%
1024x1024	4%	4%

характера шума изображения можно провести подсчитав процент идеальный пикселей от всего изображения.

В таблице 4.2 приведено сравнение характера шума для разных размеров изображения. Размер сабпикселя равен 4x4.

Таблица 4.2: Сравнение характера шума классического и квантового алгоритма избыточной выборки.

Размер изображения	Классическая выборка	Квантовая выборка
64x64	58%	88%
128x128	71%	78%
256x256	63%	75%
512x512	62%	73%
1024x1024	61%	67%

#### 4.2.4 Сравнение вычислительной сложности алгоритмов

Для того чтобы оценить время выполнения алгоритмов, нужно вывести их сложность. Вычислительная сложность алгоритма Монте – Карло напрямую зависит от размера изображения и составляет  $O(n)$  [40]. На основе изложенного алгоритма в разделе 2.4, выведем сложность выполнения квантового алгоритма избыточной выборки. При этом, примем что все квантовые операции выполняются за  $O(1)$ .



Опираясь на описание квантового алгоритма избыточной выборки из раздела 2.4, построено рассчитаем его сложность:

- перевести все пиксели холста в состояние 1 –  $O(1)$  (реализуется с помощью операции смены фазы, см. раздел 1.4.2);
- сформировать и заполнить квантовую поисковую таблицу –  $O(n * k)$ , где  $k$  – глубина итераций усиления комплексной амплитуды. Так как  $k$  – константа (см. раздел 1.4.3), мы имеем право отбросить эту константу [41], получая конечную сложность формирования таблицы  $O(n)$ ;
- посчитать количество инвертируемых пикселей (для каждого пикселя холста) –  $O(n^2)$ ;
- заполнить соответствующую ячейку карты достоверности (для всей таблицы) –  $O(n^2)$ .

Итоговая вычислительная сложность квантового алгоритма избыточной выборки составляет (4.1):

$$c * (O(1) + O(n) + O(n^2) + O(n^2)) = O(n^2) \quad (4.1)$$

где  $c$  – число цветов колоризации. Так как это число является константой (см. раздел 1.6), в конечном счете мы можем откинуть его [41].

## Вывод

Как и ожидалось, уровень зашумленности изображения примерно равен как для классической выборки, так и для квантовой. Так, например, средний процент ошибки на пиксель при размере синтезируемого изображения 512x512 для квантового алгоритма составляет 3%, а для классического 4% соответственно.

Квантовая выборка на маленьких размерах изображения выдает большой процент идеальных пикселей. Например, для изображения размером 64x64 количество идеальных пикселей составляет 88% от всего изображения. Но, с увеличением изображения этот процент незначительно падает,

и уже на размере изображения  $512 \times 512$  составляет 73%. Несмотря на это, классическая выборка даже в самом лучшем случае не добивается такого результата – ее лучший результат 71% идеальных пикселей для размера изображения  $128 \times 128$ . Квантовый алгоритм генерирует на 10%..30% идеальных пикселей больше, чем его классический аналог, из чего можно сделать вывод о сильных различиях в характере шума.

Была оценена вычислительная сложность квантового алгоритма избыточной выборки ( $O(n^2)$ ). Исходя из того факта, что сложность выборки Монте – Карло составляет  $O(n)$ , можно сделать что квантовый алгоритм будет проигрывать по времени работы классическому аналогу даже на настоящем квантовом компьютере.

# Заключение

Во время выполнения курсового проекта было реализовано программное обеспечение в котором были реализованны квантовые алгоритмы, которые в возможно применить в алгоритме трассировки лучей с целью его улучшения.

Были проанализированы и рассмотрены существующие алгоритмы трассировки лучей. В качестве той части алгоритма, которую стоит заменить на квантовую, был выбран алгоритм избыточной выборки. Данный выбор был сделан на основе анализа основных принципов построения квантовых вычислений, которые позволили выбрать наиболее подходящее решение для поставленной задачи.

В ходе выполнения поставленной задачи были получены знания в области компьютерной графики, а так же в области квантовых вычислений. Были изучены принципы эмуляции квантовых вычислений на обычных компьютерах. Поиск подходящего решения для поставленной задачи позволил повысить навыки и анализа информации.

В результате проведенной работы было получено программное обеспечение, доказывающее применимость квантовых алгоритмов в области компьютерной графики. Разработанный программный продукт реализует квантовый алгоритм избыточной выборки, который в дальнейшем возможно применить для реализации алгоритма трассировки лучей.

В ходе выполнения экспериментально-исследовательской части было установлено, что уровень шума классического алгоритма и квантового алгоритма избыточной выборки примерно одинаков. Было выявлено, что полученные изображения отличаются в характере шума: квантовый алгоритм генерирует в среднем на 15% больше идеальных пикселей чем его классический аналог. При этом, остальные пиксели крайне зашумлены, что может ускорить пост обработку изображения, например удаление и замену этих пикселей. Была оценена вычислительная сложность квантового алгоритма избыточной выборки в случае его запуска на настоящем квантовом компьютере, и был сделан не утешительный вывод что квантовый алгоритм будет выполняться медленнее, чем его классический аналог.

# Литература

- [1] The Path to Path-Traced Movies [Электронный ресурс]. Режим доступа: <https://graphics.pixar.com/library/PathTracedMovies/paper.pdf> (дата обращения: 15.11.2020).
- [2] Yuri Ozhigov, Quantum Computers Speed Up Classical with Probability Zero, Chaos Solitons Fractals 10 (1999) 1707—1714. Режим доступа: [xxx.lanl.gov/abs/quant-ph/9803064](http://xxx.lanl.gov/abs/quant-ph/9803064) (дата обращения: 15.11.2020).
- [3] Stroebel, Leslie; Zakia, Richard D. The Focal encyclopedia of photography. Focal Press. стр. 507. 1995.
- [4] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 211 – 212 с. 2019.
- [5] Joint Quantum Institute – Quantum Superposition [Электронный ресурс]. Режим доступа: <https://jqj.umd.edu/glossary/quantum-superposition> (дата обращения: 01.11.2020).
- [6] Heisenberg W. Criticisms and Counterproposals to the Copenhagen Interpretation of Quantum Theory // Physics and Philosophy: The Revolution in Modern Science. стр. 102. 2007.
- [7] Quantum computers: state of development and areas of application. [Электронный ресурс]. Режим доступа: <https://www.lead-innovation.com/english-blog/quantum-computers-state-of-development> (дата обращения: 27.11.2020).
- [8] Quantum computer emulated by a classical system. [Электронный ресурс]. Режим доступа: <https://phys.org/news/2015-05-quantum-emulated-classical.html> (дата обращения: 27.11.2020).
- [9] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 8 – 9 с. 2019.

- [10] IBM Quantum Experience - Quantum phase [Электронный ресурс]. Режим доступа: <https://quantum-computing.ibm.com/docs/iqx/guide/introducing-qubit-phase> (дата обращения: 21.09.2020).
- [11] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 107 – 123 с. 2019.
- [12] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 85 – 106 с. 2019.
- [13] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 147 – 148 с. 2019.
- [14] Rao, R. Signals and Systems. Prentice-Hall Of India Pvt. Limited, стр 152. 2008.
- [15] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 149 с. 2019.
- [16] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 223 – 227 с. 2019.
- [17] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 125 – 153 с. 2019.
- [18] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 220 – 223 с. 2019.
- [19] Programming Quantum Computers. Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia, 232 – 233 с. 2019.
- [20] Квантовый компьютер. Ю. И. Ожигов. [Электронный ресурс]. URL: [http://sqi.cs.msu.ru/files/glava\\_1.pdf](http://sqi.cs.msu.ru/files/glava_1.pdf) (дата обращения: 27.11.2020). 2017.
- [21] Квантовая физика: декогеренция. [Электронный ресурс]. URL: <https://habr.com/ru/post/449888/> (дата обращения: 10.12.2020).
- [22] Квантовые коды, исправляющие ошибки. [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/>

kvantovye-kody-ispravlyayuschie-oshibki (дата обращения: 28.11.2020). 2017.

- [23] Richard P. Feynman, Simulating Physics with Computers, International Journal of Theoretical Physics. 1982.
- [24] The Greatest Challenge of Modeling Large Quantum Computers [Электронный ресурс]. Режим доступа: <https://serokell.io/blog/modeling-large-quantum-computers> (дата обращения: 01.11.2020).
- [25] Как работают квантовые компьютеры. Собираем пазл. [Электронный ресурс]. URL: <https://habr.com/ru/post/480480/> (дата обращения: 02.12.2020).
- [26] Ivanova, O. A., «Unitary matrix», Encyclopedia of Mathematics. 2001.
- [27] Quantum supremacy using a programmable superconducting processor [Электронный ресурс]. URL: <https://www.nature.com/articles/s41586-019-1666-5> (дата обращения: 27.11.2020).
- [28] Knuth §6.2.1 («Searching an ordered table»), subsection «Binary search». 1998.
- [29] JavaScript – official site. [Электронный ресурс]. URL: <https://www.javascript.com/> (дата обращения: 01.12.2020).
- [30] QCEngine – quantum computer simulator. [Электронный ресурс]. URL: <https://oreilly-qc.github.io/docs/build/index.html> (дата обращения: 26.10.2020).
- [31] Node.js – official site. [Электронный ресурс]. URL: <https://nodejs.org/en/> (дата обращения: 01.12.2020).
- [32] GNOME Gtk+ bindings for NodeJS. [Электронный ресурс]. URL: <https://www.npmjs.com/package/node-gtk> (дата обращения: 20.11.2020).
- [33] Cairo-backed Canvas implementation for Node.js. [Электронный ресурс]. URL: <https://www.npmjs.com/package/canvas> (дата обращения: 20.11.2020).

- [34] Jest – Delightful JavaScript Testing. [Электронный ресурс]. URL: <https://jestjs.io> (дата обращения: 01.12.2020).
- [35] Kolawa, Adam; Huizinga, Dorota. Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press, стр. 75. 2007.
- [36] Kaner, Falk, Nguyen. Testing Computer Software. Wiley Computer Publishing, стр. 42. 1999.
- [37] ESLint – Pluggable JavaScript linter. [Электронный ресурс]. URL: <https://eslint.org> (дата обращения: 01.12.2020).
- [38] Visual Studio Code - Code Editing. Redefined. [Электронный ресурс]. URL: <https://code.visualstudio.com> (дата обращения: 01.12.2020).
- [39] Rebus community – Integrated Development Environment [Электронный ресурс]. Режим доступа: <https://press.rebus.community/programmingfundamentals/chapter/integrated-development-environment/> (дата обращения: 10.12.2020).
- [40] Computational complexity analysis for Monte Carlo approximations of classically scaled population processes. 2018.
- [41]