ОТЧЕТ

о выполнении НИОКР по теме:

«Разработка учебно-демонстрационного стенда для изучения  
технологических процессов изготовления деталей из полимерных  
композиционных материалов методом вакуумной инфузии»

Содержание

[1. Моделирование процесса вакуумной инфузии методом клеточных автоматов 3](#_Toc536553022)

[1.1. Вычисление давления в узлах 3](#_Toc536553023)

[1.2. Определение степени заполнения узла 5](#_Toc536553024)

[1.3. Вычисление времени шага 6](#_Toc536553025)

[1.4. Вязкости связующего 7](#_Toc536553026)

[2. Параметры, необходимые для работы алгоритма моделирования 7](#_Toc536553027)

[2.1. Сетка преформы 7](#_Toc536553028)

[2.2. Параметры наполнителей преформы 8](#_Toc536553029)

[2.3. Параметры связующего 9](#_Toc536553030)

[2.4. Параметры подачи и съёма связующего 9](#_Toc536553031)

[3. Методика измерения основных параметров 10](#_Toc536553032)

[3.1. Методика измерения вязкости 10](#_Toc536553033)

[3.2. Методика измерения температурного коэффициента 12](#_Toc536553034)

[3.3. Методика измерения проницаемости 13](#_Toc536553035)

[3.4. Методика измерения пористости 14](#_Toc536553036)

[4. Основные концепции, использованные при алгоритмизации 15](#_Toc536553037)

[4.1. Общее описание алгоритма моделирования 15](#_Toc536553038)

[4.2. Автоматическое соединение узлов различных слоёв 17](#_Toc536553039)

[4.3. Многопоточность 19](#_Toc536553040)

[4.4. Визуализация при помощи трёхмерной графики 26](#_Toc536553041)

[4.5. Формат хранения детали в файле 29](#_Toc536553042)

[5. Внутренняя структура программного обеспечения 31](#_Toc536553043)

[5.1. Основные классы 32](#_Toc536553044)

[5.2. Классы элементов пропитываемой детали, используемые в расчётах 33](#_Toc536553045)

[5.3. Классы визуализации элементов пропитываемой детали 33](#_Toc536553046)

[5.4. Классы создания сеток слоёв 34](#_Toc536553047)

[5.5. Классы сохранения и загрузки созданной детали 35](#_Toc536553048)

[5.6. Вспомогательные структуры и классы 35](#_Toc536553049)

[Литература 37](#_Toc536553050)

# 1. Моделирование процесса вакуумной инфузии методом клеточных автоматов

## 1.1. Вычисление давления в узлах

Течение связующего через армирующий наполнитель моделируется законом Дарси, который устанавливает линейное отношение между скоростью потока и градиентом прикладываемого давления:

*u* = – [*K*] / μ · *p*, (1.1)

где u – скорость потока, м/с; [K] – тензор проницаемости, м2; μ – вязкость смолы, Па · с; *p* – давление в смоле, Па;  – оператор Гамильтона.

вверх.

Также необходимо учитывать уравнение неразрывности:

 · *u* = 0. (1.2)

Подставляя (1.1) в (1.2), исключаем *u* и получаем уравнение для определения давления *p*:

 · { ( 1/μ ) [*K*] *p* } = 0. (1.3)

Уравнение (1.3) является эллиптическим уравнением второго порядка в частных производных относительно давления.

В данном продукте используется подход к имитационному моделированию, основанный на идее клеточных автоматов. Клеточные автоматы были предложены в 1940 году Джоном фон Нейманом, Станиславом Уламом и Аланом Тьюрингом.

Клеточные автоматы представляют собой пространственно дискретную систему, зависящую от времени, в которой каждое следующее состояние индивидуальной клетки через заданный период времени зависит от её собственного предыдущего состояния и предыдущих состояний конечного числа заданных соседних клеток.

В сравнении с наиболее популярным методом моделирования физических явлений, описываемых уравнениями в частных производных, методом конечных элементов, метод клеточных автоматов имеет как достоинства, так и недостатки.

Основным и наиболее существенным достоинством является меньший объём необходимых вычислений. Это приводит к сокращению времени моделирования и снижению требований к аппаратному обеспечению компьютера, на котором производится моделирование.

К недостаткам можно отнести более низкую точность. Также при моделировании с использованием клеточных автоматов для сохранения точности плотность расположения клеток (узлов сетки, на которые разбита деталь) должна быть максимально однородной. Метод конечных элементов же требует разбиения детали на треугольники, максимально близкие к равносторонним.

На основе закона Дарси была выведена формула [1] для обновления значения давления в клетке:

, (1.4)

где *pt* – давление в данной клетке на шаге *t*, Па;

*pi,t* – давление в *i*-й соседней клетке на шаге *t*, Па;

*K* - проницаемость для данной клетки, м2;

- средняя проницаемость, м2;

*d* – толщина (высота полости) для данной клетки, м;

*di* – толщина (высота полости) для *i*-й соседней клетки, м;

П – пористость для данной клетки, безразмерная;

П *i* – пористость для *i*-й соседней клетки, безразмерная;

*li* – расстояние до *i*-й соседней клетки, м;

– среднее расстояние между клетками, м;

*q*, *r*, *s* – весовые коэффициенты, приняты равными 1, 1 и 2 соответственно;

*m* – число соседей.

Все влияющие параметры, такие как проницаемость (часть *A* формулы (1.4)), рабочий объём клетки, представленный высотой полости и пористостью (часть *B*), а также расстояния между клетками (часть C) нормализуются. Часть *A* учитывает локальную проницаемость клетки. Часть *B* задаёт вес клетки относительно её соседей, в зависимости от их параметров. Часть *C* делает клеточный автомат независимым от плотности клеток, т.е. позволяет сделать поведение заполнения всегда одинаковым для клеток с одинаковыми параметрами, вне зависимости от плотности.

Параметр *m* предписывает число соседних клеток, учитываемых при расчёте давления, т.е. одно и то же число соседей используется, вне зависимости от их плотности.

Весовые коэффициенты *q*, *r* и *s* должны быть подобраны так, чтобы обеспечить наилучшую аппроксимацию поведения реального потока.

В данном алгоритме используется изотропное значение проницаемости *K*. В случае необходимости клеточный автомат может быть расширен для использования анизотропной проницаемости.

## 1.2. Определение степени заполнения узла

Изначально все клетки являются пустыми и имеют нулевое давление связующего *p*. В клетках, непосредственно соединённых с устройством подачи связующего, давление является постоянным и в случае вакуумной инфузии равно атмосферному давлению *patm*. Когда расчёт начинается, это давление распространяется через окружающие точки. В точке вакуумирования давление связующего не может подняться выше создаваемого давления вакуума *pvac*. На каждом шаге расчёта оцениваются величина давления *p* в клетках и степень её заполнения *f*. Степень заполнения *f* определяется как отношение давления связующего в клетке *p* к давлению вакуума *pvac*. Клетка считается заполненной, если в ней давление связующего *p* достигло давления вакуума *pvac* (1.5).

. (1.5)

Вычисления оканчиваются когда все клетки становятся заполненными (для всех клеток *f* = 1), либо на последнем шаге не обновилась величина давления ни для одной клетки (для всех клеток *pt* = *pt*+1).

## 1.3. Вычисление времени шага

Шаги вычисления имеет переменную длительность. Длительность *t*-го шага определяется по формуле (1.5):

, (1.6)

где τ*t* – длительность шага t, с;

μ – вязкость связующего (смолы), Па · с:

*nt* – число полностью заполненных клеток (для которых *f* = 1) на шаге *t*;

– среднее расстояние между клетками, м;

- средняя проницаемость, м2;

*patm* – давление, с которым осуществляется подвод связующего, Па;

*pvac* – давление вакуума, Па.

Точность подсчёта времени с использованием (1.6) имеет сильную зависимость от однородности плотности пространственного расположения клеток.

## 1.4. Вязкости связующего

Вязкость связующего имеет зависимость от температуры. Связь вязкости и температуры может быть выражена при помощи формулы Френкеля-Андраде [2], которое основывается на соображениях о движениях молекул:

, (1.7)

где:

μ – динамическая вязкость, Па · с;

*T* – температура, К;

*w* – константа, определяемая природой жидкости, Дж;

*A* – константа, определяемая природой жидкости, Па · с;

*k* = 1,38 · 10-23 Дж/К – постоянная Больцмана.

Формулу Френкеля-Андраде (1.7) можно преобразовать для вычисления вязкости относительно некоторой номинальной вязкости μ(*T*0) = μ0, измеренной при номинальной температуре *T*0 (в данном случае нужно принять *T*0 = 298,15 К = 25 **°**C):

,

, (1.8)

Подставив (1.8) в (1.7), получаем формулу для вычисления вязкости с учётом температурной зависимости:

. (1.9)

# 2. Параметры, необходимые для работы алгоритма моделирования

## 2.1. Сетка преформы

Сетка должна содержать набор координат узлов (клеточных автоматов), а также набор связей узлов друг с другом (для задания соседних клеточных автоматов).

Координаты задаются **в метрах**. В разработанном программном обеспечении предусмотрена функция перевода координат в метры из миллиметров при загрузке файла сетки.

При добавлении нескольких слоёв, каждый слой представляется отдельной сеткой, загружаемой из файла. Соединения между узлами различных слоёв создаются автоматически перед началом процесса моделирования.

Сетка должна быть создана заранее в сторонней программе. Существует ряд решений, как свободных, так и коммерческих, позволяющих создавать подобные сетки. Наиболее популярным программным пакетом с открытым исходным кодом для создания сеток является Gmsh.

Разработанное программное обеспечение для импорта сеток слоёв использует файлы формата .MSH. Данный формат был создан для хранения сеток разработчиками программы Gmsh.

Структура файлов формата:

$MeshFormat

*[номер версии]* 0 *[размер числа с плавающей точкой]*

$EndMeshFormat

$Nodes

*[число слоёв в сетке]*

*[номер слоя] [x-координата] [y-координата] [z-координата]*

...

$EndNodes

$Elements

*[число элементов]*

*[номер элемента] [тип элемента] [число тегов] <теги> [список узлов, с которыми связан элемент]*

...

$EndElements

## 2.2. Параметры наполнителей преформы

Преформа может состоять из множества слоёв из различных материалов. Используются следующие параметры:

* **Толщина** – толщина ткани материала. Задаёт высоту полости в клетке.

[*d*] = [м]

* **Проницаемость** – параметр, характеризующий свойство вещества пропускать сквозь себя жидкости и газы при наличии перепада давления.

[*K*] = [м2]

* **Пористость** – доля объёма пор в общем объёме пористого тела. Является безразмерной величиной от 0 до 1. 0 соответствует материалу без пор; пористость 1 недостижима. Пористость равна отношению свободного объёма *Vc*, не заполненного элементарными структурными частицами, к общему объёму *V* тела:

*,*

[П] = [м3] / [м3] = []

где *VТ* — объём твёрдого скелета или матрицы.

## 2.3. Параметры связующего

Связующее характеризуется номинальной вязкостью и температурным коэффициентом.

* **Номинальная вязкость** – динамическая вязкость жидкости, измеренная при температуре 25 °C (298,15 К).

[μ0] = [Па · с]

* **Температурный коэффициент** – число, используемое для определения вязкости жидкости при заданной температуре. Измеряется в Паскалях на секунду (Па · с).

[*A*] = [Па · с]

## 2.4. Параметры подачи и съёма связующего

Источник связующего определяется положением на детали, диаметром трубки и давлением подачи.

* **Диаметр источника** – диаметр трубки, подключённой к преформе, по которой подаётся связующее.

[*Dinj*] = [м]

* **Давление подачи** – давление, с которым осуществляется впрыск связующего. Для процесса вакуумной инфузии соответствует атмосферному давлению. По умолчанию принимается *patm* = 105 Па.

[*patm*] = [Па]

Источник связующего определяется положением трубки откачки на детали, диаметром этой трубки и давлением вакуумирования.

* **Диаметр источника** – диаметр трубки, подключённой к преформе, по которой подаётся связующее.

[*Dvac*] = [м]

* **Давление вакуума** – давление, создаваемое компрессором в рабочей области детали. Величина больше нуля и меньше атмосферного давления.

[*pvac*] = [Па]

# 3. Методика измерения основных параметров

## 3.1. Методика измерения вязкости

Динамическая вязкость – характеристика вещества, численно равная силе трения, возникающей между двумя слоями жидкости площадью по 1 м2 каждый при градиенте скорости, равном 1 м/с на метр. Размерность динамической вязкости [µ] = [Па·с].

Коэффициент динамической вязкости зависит от природы жидкости и для жидкости с повышением температуры уменьшается.

Для измерения вязкости (вискозиметрии) применяют ряд экспериментальных методов. Измерить динамическую вязкость при определённой температуре можно при помощи **метода Стокса** [3].

Для определения вязкости жидкости по методу Стокса берётся высокий цилиндрический сосуд с исследуемой жидкостью (рис. 3.1). На сосуде имеются две кольцевые метки **А** и **В**, расположенные на расстоянии *l* друг от друга. Уровень жидкости должен быть выше верхней метки на *l*0 = 4…5 см, чтобы к моменту прохождения шарика мимо верхней метки его скорость можно было считать установившейся.

Бросая шарик с радиусом *r* и плотностью ρ в сосуд, наполненный исследуемой жидкостью с известной плотностью ρ0, отмечают по секундомеру время *t* прохождения шариком расстояния *l* = **АВ** между двумя метками.

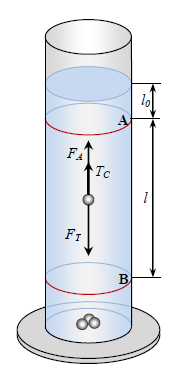


Рис. 3.1. Лабораторная установка для измерения вязкости

При падении шарика радиусом *r* в цилиндрической трубе радиусом *R*0, высотой *h* с учётом влияния границ формула Стокса преобразуется к виду:

, (3.1)

*,*

где:

*g* = 9,8 м/с2 – ускорение свободного падения;

*r* – радиус шарика, м;

*R*0 – радиус сосуда, м;

*h* – высота сосуда, м;

ρ – плотность шарика, кг/м3;

ρ0 – плотность шарика, кг/м3;

*l* – расстояние между метками сосуда, м;

*t* – время прохождения шариком расстояния между метками, с.

Таким образом, зная плотности материала шарика и жидкости, радиусы шарика и сосуда, скорость установившегося движения шарика, по формуле (3.1) можно вычислить динамическую вязкость жидкости.

**Выполнять измерение динамической вязкости следует при температуре, соответствующей стандартному состоянию, – 25 °C (298,15 К).**

## 3.2. Методика измерения температурного коэффициента

Вязкость связующего имеет зависимость от температуры. Связь вязкости и температуры, выведенная из формулы Френкеля-Андраде, определяется выражением (1.9).

Значение температурного коэффициента *A* можно получить, произведя измерения вязкостейμ0 и μ1 при двух различных температурах *T*0 и *T*1 соответственно и решив систему из двух уравнений (1.9):

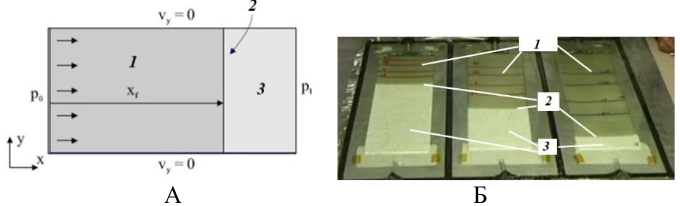
,

[.

## 3.3. Методика измерения проницаемости

В общем случае, когда среда является анизотропной, проницаемость характеризуется тензором [*K*]. В [4] приведён одномерный канальный метод, позволяющий определить плоскостные компоненты тензора проницаемости в главной системе координат. В случае изотропной среды достаточно определить только одну компоненту тензора.

Эксперимент проводится путём осуществления вакуумной инфузии прямоугольных образцов. Смола поступает с одного края образца через линейный источник, на противоположной стороне поддерживается постоянный уровень вакуума. Экспериментально найденная зависимость координаты фронта от времени подставляется в аналитическое выражение для проницаемости. На рис. 3.2 представлена схема эксперимента, на рис. 3.2Б – пример экспериментальной установки. На рис. 3.2 введены следующие обозначения: 1 - пропитанная зона, 2 – фронт пропитки, 3 – непропитанная зона.

Рис. 3.2. Схема одноканального метода (А) и экспериментальная установка (Б). 1 – Пропитанная зона. 2 – Фронт пропитки. 3 – Непропитанная зона

Проницаемость вдоль направления *x* определяется по следующей формуле:

*,*

[*Kx*] = [ (Па · с · м2) / (Па · с) ] = [м2],

где:

*t0* - момент времени начала измерения, с;

*tf* - момент времени окончания измерения, с;

∆*t* = *tf - t0* - время измерения процесса, с;

*xf*(*t*) - координата фронта пропитки (начало координат совпадает с положением источника), м;

µ - вязкость смолы, Па·с;

∆*p* - разность давлений на источнике впуска связующего и давления на фронте, Па:

∆*p = patm - pf ,*

*patm* - давление в источнике впуска связующего, Па;

*pf* - давление вакуумирования (давление на фронте течения), Па.

## 3.4. Методика измерения пористости

Методы исследования пористых структур описаны в [5].

Пористость материала, не содержащего закрытых пор (не сообщающихся с поверхностью тела), можно измерить следующим образом:

1) Измерить габариты образца и по результатам измерений вычислить его общий объём *V.*

2) Поместить образец в жидкостный волюмометр (сосуд, проградуированный в единицах объёма). Рабочая жидкость должна обладать хорошей смачивающей способностью и не взаимодействовать с телом. Объём скелета образца *VТ* определяется по объёму вытесненной им жидкости.

3) По полученным значениям вычислить величину пористости по формуле:

.

# 4. Основные концепции, использованные при алгоритмизации

## 4.1. Общее описание алгоритма моделирования

Описанный выше метод моделирования был реализован на языке C++ с использованием возможностей стандарта C++11.

В разработанной программе каждый узел (клеточный автомат) хранится в отдельном объекте класса *VSimNode*. В узле хранится следующая информация:

* *x*, *y*, *z* – координаты узла, м;
* текущее значение давления, Па;
* значение давления, рассчитанное для следующего шага, Па;
* тип узла: обычный, точка подвода связующего, точка съёма связующего;
* список указателей на соседние узлы;
* указатель на объект, хранящий параметры материала (структура *VCloth*).

Поверхность трёхмерной модели разбивается на треугольники. Информация о треугольниках хранится в объектах класса *VSimTriangle*. Треугольник содержит указатели на узлы, являющиеся его вершинами. Информация о степени заполнения элемента поверхности, представленного треугольником, характеризуется цветом треугольника. Цвет треугольника характеризует среднюю степень заполнения узлов, находящихся в его вершинах. Шкала цветов представлена на рис. 4.1.



Рис. 4.1. Шкала цветовых обозначений степени пропитки

За процесс симуляции отвечает объект класса *VSimulator*. Он содержит указатели на все активные узлы и треугольники, параметры симуляции (класс *VSimulationParameters*), информацию о протекании симуляции (структура *VSimulationInfo*).

На каждом шаге процесса симуляции, происходящего в объекте класса *VSimulator*, осуществляются следующие действия:

1. расчёт длительности последнего шага τ по формуле (1.6) , с;
2. измерение времени, ушедшего на моделирование последнего шага τм, с;
3. расчёт фактора реального времени: τ / τм;
4. определение текущей степени пропитки детали, %;
5. определение среднего давления связующего в детали, Па;
6. вычисление новых величин давления для каждого из узлов по формуле (1.4);
7. обновление значения давления в каждом из узлов;
8. определение, обновилось ли значение давления хотя бы в одном из узлов;
9. определение количества полностью заполненных узлов *n*, используемого при подсчёте времени по формуле (1.6);
10. обновление цветов треугольников;
11. оповещение объекта графической визуализации (*VGraphicsViewer*) о том, что был произведён новый шаг симуляции.

Если за шаг не обновилось значение давления ни для одного из узлов, либо все узлы заполнились, симуляция останавливается.

## 4.2. Автоматическое соединение узлов различных слоёв

Одной из задач, возникшей при разработке программы имитационного моделирования, является соединение (добавление в соседи) близлежащих узлов различных слоёв. Необходимо это для обеспечения взаимного влияния процессов пропитки в различных слоях.

Для сохранения точности расчётов при использовании метода клеточных автоматов, необходимо соединять только те узлы, расстояние между которыми не сильно отличается от среднего расстояния между узлами внутри слоя. Было принято решение соединять узлы, расстояние между которыми находится в диапазоне ±10% относительно среднего расстояния между узлами одного из соединяемых слоёв.

Очевидным подходом к выполнению соединения узлов является их последовательный перебор. Необходимо перебрать все узлы первого слоя и для каждого из них вычислить расстояние для каждого из узлов второго слоя. Т.е. для каждого из узлов первого слоя перебираются все узлы второго слоя. Таким образом, сложность вычислений составляет *O*(*n* · *m*), где *n* и *m* – число узлов на первом и на втором слое соответственно. При большой степени детализации число узлов на каждом из слоёв может достигать десятков тысяч. В этом случае для соединения слоёв потребуется длительное время.

Для решения данной проблемы было решено создать структуру данных с константной сложностью поиска *O*(1) узлов по координатам. Такая структура данных была реализована в классе *VNodesVolume*.

В основе реализации данного класса лежит построение вокруг слоя описывающего параллелепипеда, который разбивается на кубические ячейки одинакового размера. В каждой ячейке содержится список указателей на узлы, координаты которых попадают в данную ячейку. Такой параллелепипед хранится в трёхмерном массиве.

В таком массиве при поиске узлов можно не перебирать все элементы, а просматривать только те, которые находятся в ячейках, удовлетворяющих заданным требованиям (рис. 4.2).

При соединении слоёв рассматриваются узлы только из тех ячеек, которые попадают в пространство между двумя сферами радиусами и . Центр сфер – координата узла другого слоя, с которым необходимо выполнить соединение. Количество рассматриваемых ячеек не зависит от общего числа узлов в слое.

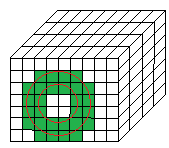


Рис. 4.2. Выбор списков узлов, потенциально удовлетворяющих критерию

При использовании данного алгоритма сложность соединения слоёв составляет *O*(*n*). Также *O*(*n*) составляет сложность построения параллелепипеда, содержащего списки указателей на узлы.

Недостатком является увеличение объёма потребляемой оперативной памяти. Для обеспечения максимального быстродействия требуется подбирать такой размер стороны ячейки, чтобы в каждую ячейку попадало по одному узлу, и при этом число пустых ячеек было минимальным. Также при возрастании числа ячеек возрастает размер потребляемой памяти.

Пустой объект класса список на 64-х разрядном компьютере занимает 24 байта. Таким образом, например, для слоя с габаритами 2 м x 4 м x 0,04 м при создании ячеек со стороной 0,01 м потребуется выделить 7,32 МиБ памяти.

Эмпирическим путём было получено, что наилучшее значение размера стороны ячейки составляет .

## 4.3. Многопоточность

Процесс моделирования может занимать длительное время. Так как подавляющее большинство современных компьютеров оснащено многоядерными процессорами, если требуется, чтобы программа выигрывала от увеличения вычислительной мощности, то её необходимо проектировать как набор параллельных задач [6].

Параллелизм – это одновременное выполнение двух или более операций. Наиболее применяемый в наше время подход к реализации параллелизма – запуск нескольких потоков в одном процессе (многопоточность).

Многопоточность – свойство платформы, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени.

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Любой выполняющийся процесс имеет как минимум один (главный) поток.

Существует два способа применить распараллеливание для повышения производительности [6]. Первый подход – *распараллеливание по задачам*. Он заключается в разбиении задачи на части и их параллельном запуске, в результате чего уменьшается общее время её выполнения. Трудности при реализации этой процедуры на практике возникают из-за наличия многочисленных зависимостей между разными частями.

Второй подход – *распараллеливание по данным*. В этом случае каждый поток выполняет одну и ту же операцию, но с разными данными. Для обработки одной порции данных требуется столько же времени, сколько и раньше, но за фиксированное время можно обработать больше данных. Очевидно, что и у этого подхода есть ограничения, и не во всех случаях он дает выигрыш, но достигаемое повышение производительности иногда открывает новые возможности.

В C++11 работа с потоками осуществляется по средствам класса *std::thread*. Функция, подаваемая в качестве аргумента конструктору *std::thread*, начинает свое исполнения сразу по окончании работы конструктора. Завершение потока происходит по завершении работы исполняемой функции. После того, как объект *std::thread* создан возможны три варианта развития событий:

1. Другой поток исполнения будет ожидать завершения исполнения созданного потока. Для этого в ожидающем потоке осуществляется вызов *thread.join()*. Этот вызов блокирует вызывающий поток.
2. Главный поток исполнения может завершится до того как будет завершён созданный поток. Блокировки вызывающего потока происходить не будет. Для этого осуществляется вызов *thread.detach().*
3. В случае, если не было выполнено ни то, ни другое, при вызове деструктора объекта thread произойдёт аварийное завершение..

Использование join является более предпочтительным, так как detach представляет меньше контроля за исполнением.

В классе thread, также существует статическая функция std::thread::hardware\_concurrency, которая может вернуть количество потоков, которые могут выполняться параллельно на аппаратном уровне.

Когда появляется возможность параллельного исполнения кода, очень часто возникает необходимость конкурентного доступа к разделяемому ресурсу. Для получения предсказуемого результата от операции над разделяемым ресурсом, необходимо упорядочивать доступ к оному. Существует несколько программных инструментов, помогающих защитить разделяемые данные и сделать код потокобезопасным. Их называют примитивами синхронизации. Наиболее распространенными являются следующие примитивы: мьютексы, семафоры, условные переменные и спин-блокировки. Все они являются вариантами одной и той же концепции: они защищают кусок кода, давая только определенному потоку право получать доступ к данным и блокируя остальные.

Наиболее распространенным (и используемым) примитивом является мьютекс. Мьютекс расшифровывается как «Взаимное исключение». Это гарантия, что только один поток может выполнять код.

С++11 предоставляет 3 типа операций над базовыми мьютексами:

* *lock –* если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший *lock*, становится его обладателем. Если же некий поток уже владеет мьютексом, то текущий поток (который пытается овладеть им) блокируется до тех пор, пока мьютекс не будет освобожден и у него не появится шанса овладеть им.
* *try\_lock -* если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший *try\_lock*, становится его обладателем и метод возвращает *true.* В противном случае возвращает *false*. *try\_lock* **не** блокирует текущий поток.
* *unlock –* освобождает ранее захваченный мьютекс.

В C++11 представлено четыре вида мьютексов:

* *std::mutex* – базовый *mutex*, которым может владеть один поток в единицу времени. При попытке повторного овладения *мьютексом* потоком, уже владеющим им, произойдёт взаимная блокировка (*deadlock)* или будет брошено исключение с кодом ошики  *resource\_deadlock\_would\_occur*.
* *std::recursive\_mutex* – обладает теми же свойствами, что и *std::mutex*, но позволяет рекурсивное овладение мьютексом, т.е. многократный вызов метода *lock()* в потоке, который владеет мьютексом. При этом, метод *unlock()* должен быть вызван не меньшее количество раз, чем был вызван *lock().* В противном случае произойдёт взаимная блокировка, т.к. этот поток никогда не освободит *мьютекс* и остальные потоки будут находиться в вечном ожидании.
* *std::timed\_mutex –*обладая свойствами *std::mutex, std::timed\_mutex,* также*,* обладает дополнительными методами позволяющими блокировку на время.
* std::recursive\_timed\_mutex – рекуррентная версия *std::timed\_mutex.*

Для облегчения работы с мьютексами в C++11 существуют средства для применения для них принципа «**Получение ресурса есть инициализация» (RAII).** Эти средства реализованы в std::lock\_guard. При создании объекта lock\_guard захватывается мьютекс, переданный ему в конструкторе. В деструкторе, же, происходит освобождение мьютекса.

Ещё одним блокирующим примитивом в C++11 является условная переменная – *std::condition\_variable*. Привычным шаблоном использования подобного примитива является ожидание одного потока, наступления события в другом потоке. Этот примитив можно рассматривать как некий сигнал, появления которого необходимо ожидать.

Основными методами *condition\_variable* являются:

* *wait –* ставит поток в ожидание сигнала. Ожидание не лимитировано временем. Может принимать в качестве аргумента предикат, от результата которого будет зависеть выход потока из ожидания. Т.е. если даже *wait* был завершен благодаря сигналу, происходит проверка предиката после чего поток снова становится в ожидание, если предикат ложен. Нужно это, в первую очередь, для того, чтобы избежать реагирования на *фальшивое* пробуждение.
* *wait\_for –* ожидание лимитировано согласно аргументу.
* *wait\_until* - ожидание лимитировано согласно аргументу.
* *notify\_one –*посылает сигнал одному из ожидающих потоков; т.е. разблокирует один поток. Какой поток будет разбужен – не известно. Гарантировано лишь то, что один из них будет.
* *notify\_all –*посылает сигнал всем ожидающим потокам; т.е. разблокирует все потоки ожидающие на данном объекте *condition\_variable.*

В разработанной программе для сообщения другим потокам информации о наступлении событий был создан класс *VNotify*, реализующий обёртку над *std::condition\_variable.*

Также в C++11 реализованы атомарные переменные *std::atomic*. Они характеризуются тем, что как только начинается процесс их чтения или записи, ничто не может его прервать и ничто не может произойти с этой переменной в середине.

В разработанной программе были применены оба подхода к распараллеливанию.

Распараллеливание по задачам реализовано за счёт одновременного расчёта данных и их визуализации на трёхмерной модели. После каждой итерации расчёта объекту, отвечающему за визуализацию, при помощи условной переменной отправляется сигнал о завершении итерации расчёта. Если объект визуализации в данный момент уже завершил отрисовку результатов предыдущего шага расчёта, он к отрисовке результатов нового. Если итерация расчётов занимает в несколько меньше времени, чем итерация отрисовки, визуализация будет происходить не для каждого шага. Данные о визуализируемых цветах (массив элементов *VSimTriangle*) защищаются мьютексом.

Распараллеливание по данным реализовано за счёт одновременно расчёта новых давлений и степеней заполнения для различных частей узлов. Для этого массив указателей на узлы разбивается на приблизительно равные части, расчёт которых осуществляется параллельно. Число параллельно рассчитываемых частей равно числу возможных выполняемых потоков на аппаратном уровне минус единица (но не менее одной части).

Было проведено сравнение результатов выполнения многопоточного кода и однопоточного. Для этого было измерено время моделирования пропитки трёх деталей (табл. 4.1). Использовались следующие параметры:

* давление подачи связующего: 105 Па;
* давление вакуума: 103 Па;
* вязкость связующего при 25°C: 0,1 Па·с;
* температура: 25°C;
* проницаемость 2·10-9м2;
* толщина (высота полости): 0,002 м;
* пористость: 0,5.

Таблица 4.1.

Измерение времени моделирования с использованием многопотончости и без

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя файла | Число узлов | Число треугольников | Габариты, *x, y, z*, м | Диаметр источника, м | **Время однопоточного моделирования, с** | **Время многопоточного моделирования, с** |
| gmsh\_simple.msh | 5589 | 10832 | 2; 4; 0,04 | 0,1 | **492** | **25** |
| 1x0.5m.msh | 6764 | 13226 | 1; 0,5; 0 | 0,05 | **256** | **12** |
| paddel\_einfach.db | 526 | 949 | 0,22; 0,98; 0,05 | 0,02 | **8** | **2** |

По таблице 4.1 видно, что использование многопоточности позволяет добиться в десятки раз большей производительности, особенно при использовании слоёв с большим количеством узлов. Во многом столь большая разница в производительности обусловлена тем, что при однопоточном выполнении визуализация результата осуществлялась на каждом шаге, а при многопоточном часть шагов визуализации пропускалась.

## 4.4. Визуализация при помощи трёхмерной графики

Для визуализации результатов моделирования требуется отображать на трёхмерной сцене слои, состоящие из множества окрашенных треугольников. Для этой цели применяется графический движок. Для взаимодействия с графическим движком и управления процессом отображения элементов на экране был создан класс *VGraphicsViewer*.

Для управления отображением каждого слоя был создан класс *VGraphicsLayer*. Он включает в себя указатели на элементы, управляющие отображением каждым треугольником и узлом (отображается как небольшой кубик).

За управление отображением треугольников отвечает класс *VGraphicsTriangle*. Он содержит указатель на объект класса *VSimTriangle*, имеет те же координаты вершин и при необходимости обновляет свой отображаемый цвет на цвет, записанный в объекте класса *VSimTriangle*.

За управление отображением узлов отвечает класс *VGraphicsNode*. Он содержит указатель на объект класса *VSimNode* и отображает небольшой куб в позиции, соответствующей данному узлу.

В графических движках отображаемые на экране элементы описываются **графом сцены.**

Граф сцены представляет структуру, которая содержит логическое и зачастую пространственное представление графической сцены. Граф сцены представляет собой набор узлов такой структуры, как граф или дерево. Узел дерева (в предельной структуре дерева графа сцены) может иметь множество потомков, но зачастую только одного предка, причём действие предка распространяется на все его дочерние узлы; эффект действия, выполненного над группой, автоматически распространяется на все её элементы. Во многих программах ассоциирование матрицы преобразования на уровне любой группы и умножение таких матриц представляет собой эффективный и естественный способ обработки таких действий. Общей особенностью, к примеру, является способность группировать связанные формы/объекты в составной объект, который можно перемещать, трансформировать, выбирать и т.д. так же просто, как и одиночный объект.

Узлы в графе считываются сверху вниз и слева направо. Каждый узел, в зависимости от своего типа определяет, каким образом будет нарисована сцена. Узлы графа наследуют своё состояние от посещённых ранее узлов.

Граф сцены, используемой для визуализации процесса пропитки, представлен на рис. 4.3.

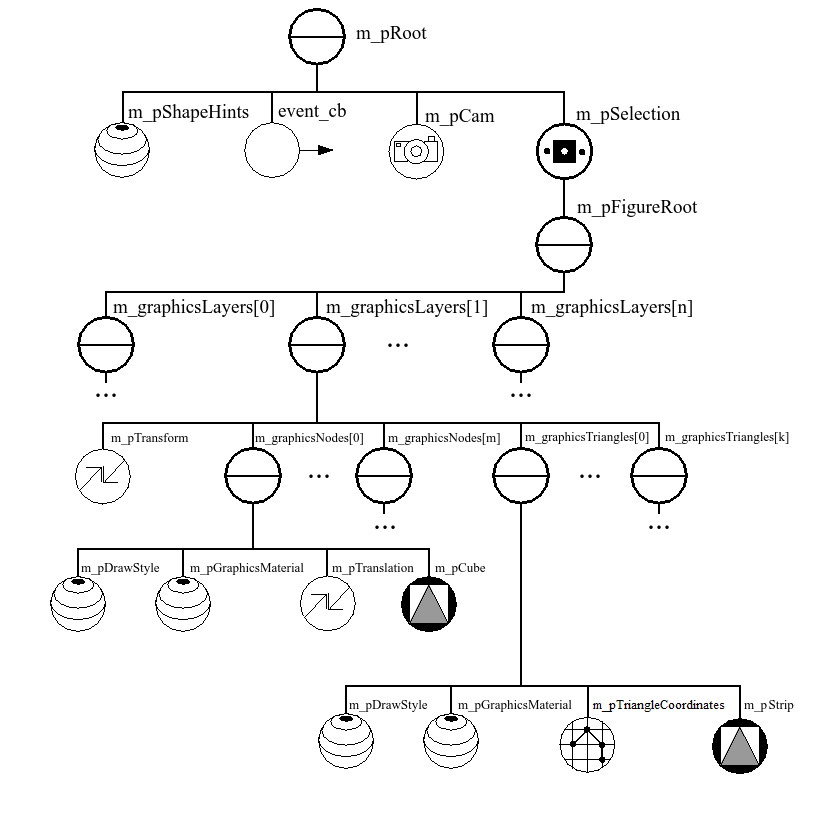


Рис. 4.3. Граф сцены для визуализации процесса пропитки

Используемые на рис. 4.3 обозначения типов узлов:

 – разделитель. Сохраняет состояние пред посещением своих дочерних элементов, а после их обхода вновь восстанавливает его.

 – отображение. Определяет внешний вид элементов.

 – функция обратного вызова. Задаёт функцию, которая будет вызвана в результате реакции сцены на событие.

 – камера. Задаёт способ отображения (перспективная или ортографическая проекции) и позицию обзора сцены.

 – выбор. Позволяет пользователю при помощи мыши выбирать дочерние узлы.

 – трансформация. Задаёт пространственное положение узла при помощи матрицы трансформации.

 – фигура. Определяет геометрию отображаемого объекта.

 – топология. Определяет координаты в пространстве.

## 4.5. Формат хранения детали в файле

В программе присутствует возможность сохранения обрабатываемой детали в файл с возможностью последующей загрузки и продолжения моделирования. Для хранения данных применяется язык разметки XML.

Стандарт XML был впервые опубликован Консорциумом Всемирной паутины в 1998 году. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. *Расширение XML*— это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов. Преимуществами XML являются сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов.

Для реализации сохранения и загрузки детали из файла было разработано три класса. Используемые теги хранятся в классе *VModelImportExport*. За сохранение в файл отвечает класс *VModelExport*. За загрузку из файла отвечает класс *VModelImport*.

В сохраняемый файл записываются:

* информация о состоянии симуляции: время процесса, время моделирования, фактор реального времени, процент выполнения задачи, среднее давление связующего, номер итерации;
* параметры моделирования: диаметр трубки подачи связующего, диаметр трубки съёма связующего, температура, давление подачи, давление вакуума, коэффициенты *q*, *r*, *s*, средняя дистанция между клетками, средняя проницаемость, число узлов,
  + характеристики связующего: температурный коэффициент, номинальная вязкость, имя материала;
* находится симуляция на паузе или была остановлена;
* ограничение на время моделирования, необходимо ли его ограничивать;
* список слоёв:
  + характеристики слоя: используется ли при расчётах, отображается ли на экране, минимальный идентификатор узла в слое, максимальный идентификатор узла в слое, минимальный идентификатор треугольника в слое, максимальный идентификатор треугольника в слое, число узлов, число треугольников,
    - характеристики материала слоя: высота полости, проницаемость, пористость, цвет (до пропитки), имя,
    - список узлов:
      * характеристики узла: идентификатор, давление, роль (обычный, связан с устройством подачи, связан с устройством снятия связующего), *x*, *y*, *z* координаты,
    - список треугольников:
      * характеристики треугольника: идентификатор, цвет, три идентификатора узлов, находящихся в вершинах треугольника;
* список соединений узлов:
  + характеристики соединения: идентификатор узла, список идентификаторов соседних узлов.

# 5. Внутренняя структура программного обеспечения

Модуль весь алгоритм моделирования реализован в модуле *v\_sim*. Диаграмма классов данного модуля представлена на рис. 5.1.

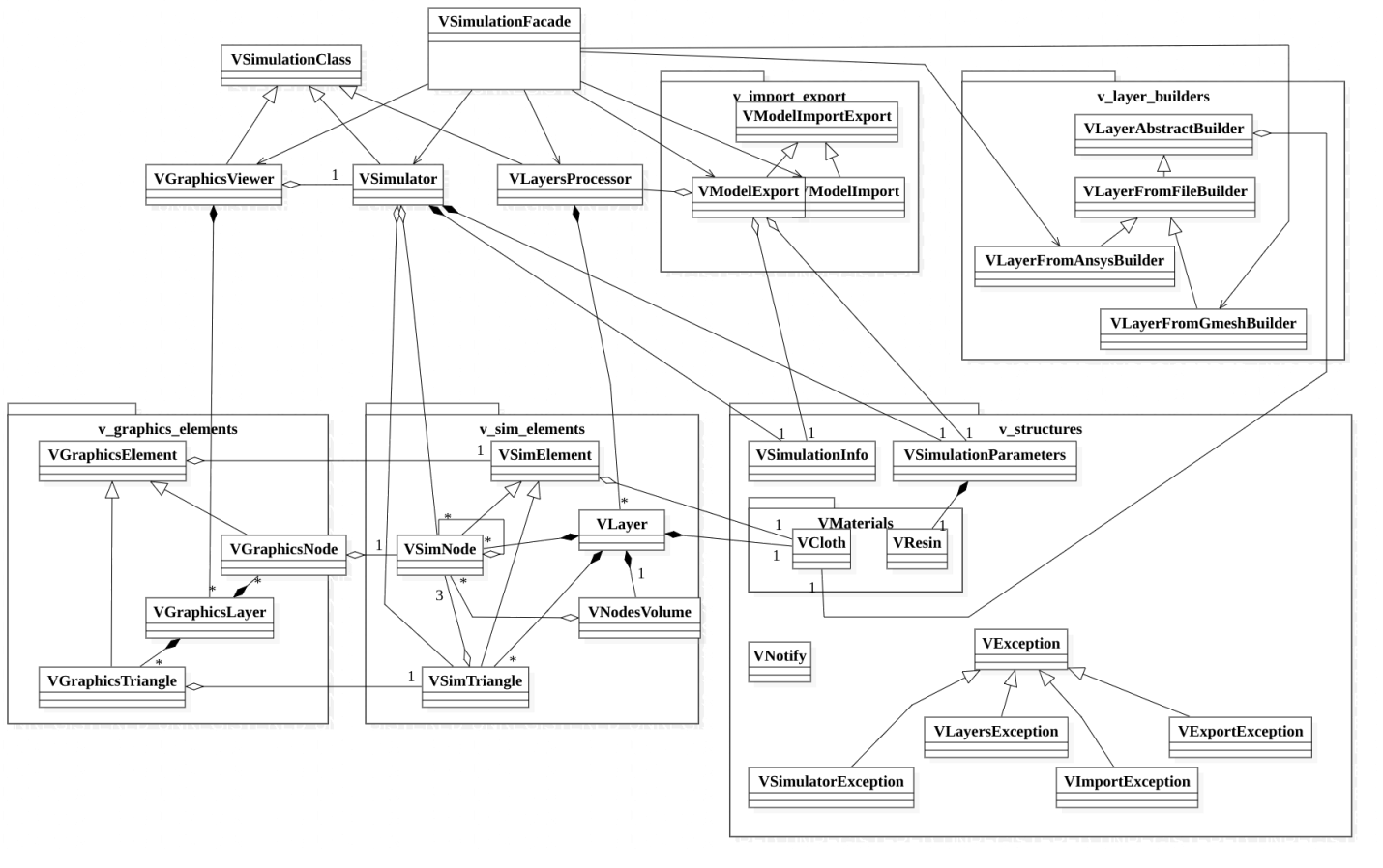


Рис. 5.1. Диаграмма классов программы моделирования

## 5.1. Основные классы

Основные классы, реализующие процесс моделирования:

***VSimulationFacade*** – класс, обеспечивающий внешний интерфейс взаимодействия со всей системой моделирования и организующий её работу.

***VSimulationClass*** – базовый класс для классов *VSimulator*, *VGraphicsViewer*, *VLayersProcessor*, отвечающих за моделирование. Реализует функции для работы с мьютексами для защиты наборов узлов и треугольников, общие для этих трёх классов.

***VSimulator*** – основной класс. Реализует алгоритм моделирования, описанный в пункте 4.1.

***VGraphicsViewer*** – класс, реализующий визуализацию процесса пропитки. Строит трёхмерную сцену, описанную в пункте 4.4.

***VLayersProcessor*** – класс для работы со слоями. Содержит в себе набор всех слоёв. Предоставляет интерфейсы для добавления, удаления, включения, отключения, отображения, скрытия, соединения слоёв; для определения габаритов детали; для указания на слоях позиций подвода и съёма связующего.

## 5.2. Классы элементов пропитываемой детали, используемые в расчётах

Классы, характеризующие структурные единицы пропитываемой детали, используемые в процессе расчёта, хранятся в модуле *v\_sim\_elements*. Модуль содержит следующие классы:

***VSimElement*** – абстрактный базовый класс, определяющий основные методы взаимодействия с элементарными структурными единицами, на которые разбивается пропитываемая деталь: узлами (класс *VSimNode*) и треугольниками (класс *VSimTriangle*).

***VSimNode*** – класс, созданный для хранения информации, характеризующей узел моделирования (клеточный автомат) на текущей итерации. Подробнее об информации, хранящейся в экземплярах данного класса, написано в пункте 4.1.

***VSimTriangle*** – класс, созданный для хранения информации о треугольниках, на которые разбивается трёхмерная деталь (см. пункт 4.1).

***VLayer*** – класс, объекты которого позволяют манипулировать слоями в целом. Содержит в себе указатели на все узлы и все треугольники слоя.

***VNodesVolume*** – структура данных, хранящая указатели на узлы в ячейках трёхмерного массива (представляющего параллелограмм). Содержит методы для быстрого поиска узлов в заданном геометрическом месте точек (см. пунк 4.2).

## 5.3. Классы визуализации элементов пропитываемой детали

Классы, отвечающие за графическое отображение структурных единиц пропитываемой детали, хранятся в модуле *v\_graphics\_elements*. Данные классы реализуют подграфы графа сцены (пункт 4.4). Модуль содержит следующие классы:

***VGraphicsElement*** – абстрактный базовый класс, определяющий основные свойства отображения элементов детали: узлов (класс *VGraphicsNode*) и треугольников (класс *VGraphicsTriangle*).

***VGraphicsNode*** – класс, определяющий параметры отображения узлов. Узлы отображаются в виде небольших кубов. Объекты данного класса содержат указатели на соответствующие им объекты класса *VSimNode*.

***VGraphicsTriangle*** – класс, определяющий параметры отображения треугольников. Объекты данного класса содержат указатель на соответствующие им объекты класса *VSimTriangle*.

***VGraphicsLayer*** – класс, определяющий параметры отображения слоя в целом. Объекты данного класса содержат указатели на весе объекты классов *VGraphicsNode* и *VGraphicsTriangle*, относящихся к данному слою.

## 5.4. Классы создания сеток слоёв

Классы, отвечающие за создание сеток слоёв (см. пункт 2.1), хранятся в модуле *v\_layer\_builders*. На данный момент реализовано только создания сеток слоёв за счёт их импорта из файлов. Классы модуля:

***VLayerAbstractBuilder*** – абстрактный базовый класс, определяющий интерфейс класса, создающего слой.

***VLayerFromFileBuilder*** – класс, содержащий основные методы, необходимые для импорта сетки слоя из файла, вне зависимости от формата файла.

***VLayerFromGmeshBuilder*** – класс, содержащий методы, необходимые для импорта сетки слоя из файла формата *Gmsh*.

***VLayerFromAnsysBuilder*** – класс, содержащий методы, необходимые для импорта сетки слоя из файла формата *Ansys*.

## 5.5. Классы сохранения и загрузки созданной детали

Классы, отвечающие за сохранение пропитываемых деталей в файл и их последующую загрузку (см. пункт 4.5), хранятся в модуле *v\_import\_export*. Классы модуля:

***VModelImportExport*** – абстрактный базовый класс, описывающий XML-теги, используемые при сохранении и загрузке детали.

***VModelExport*** – класс, содержащий методы, необходимые для сохранения модели в файл.

***VModelImport*** – класс, содержащий методы, необходимые для загрузки сохранённой модели из файла.

## 5.6. Вспомогательные структуры и классы

Вспомогательные классы хранятся в модуле *v\_structures*. Классы (структуры) модуля:

***VSimulationInfo*** – структура, хранящая информацию о текущем состоянии процесса моделирования. Информация включает: прошедшее время пропитки, время моделирования, фактор реального времени, процент выполнения задачи, среднее давление, номер итерации.

***VSimulationParameters*** – класс, хранящий текущие параметры, необходимые для моделирования. Такими параметрами являются: диаметр трубки подачи связующего, диаметр трубки съёма связующего, температура, давление подачи, давление вакуума, коэффициенты *q*, *r*, *s*, средняя дистанция между клетками, средняя проницаемость, число узлов, характеристики связующего (экземпляр структуры *VResin*).

***VCloth*** – структура, хранящая информацию о параметрах материала наполнителя. Информация включает: проницаемость, высоту полости, пористость, цвет (которым обозначается слой до начала пропитки), имя материала.

***VResin*** – структура, хранящая информацию о параметрах материала связующего. Информация включает: номинальную вязкость (вязкость при температуре 25 °C), температурный коэффициент, имя материала.

***VNotify*** – класс, созданный для оповещения других потоков о произошедших событиях (см. пункт 4.3).

***VException*** – базовый класс исключения, вырабатываемого модулем симуляции.

***VSimulatorException*** – класс исключения, возникающего при некорректном изменении модели в то время, как запущен процесс симуляции.

***VLayersException*** – класс исключения, возникающего при некорректном изменении конфигурации слоёв детали.

***VImportException*** – класс исключения, возникающего в результате ошибки импорта информации из файла.

***VException*** – класс исключения, возникающего в результате ошибки сохранения информации в файл.

# Литература

1. Henne M., Barandun G. A. Simulation of LCM processes using cellular automats //The 10th International Conference on Flow Processes in Composite Materials (FPCM10) – 2010.

2. Френкель Я. И. Кинетическая теория жидкостей. Наука. – 1975.

3. Никулин С.С., Чех А.С. Определение вязкости жидкости методом Стокса: методические указания. – Тамбов: Изд-во ГОУ ВПО ТГТУ, 2011.

4. Щеглов Б. А., Сафонов А. А. Теоретические основы и прикладные задачи технологии композитов. – 2015.

5. Фандеев В. П., Самохина К. С. Методы исследования пористых структур //Интернет-журнал Науковедение. – 2015. – Т. 7. – №. 4 (29).

6. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. – Litres, 2017.