# Разработка программ для моделирования комнаты Эймса

В результате работы над проектом было написано несколько программ на языке Python:

* Ames.py – позволяет получить выкройки пола, потолка и стен комнаты Эймса в векторном формате SVG (Scalable Vector Graphics). При этом имеется возможность задать размеры комнаты, положение точки зрения, глубину перспективного искажения, а также задать положение плоских элементов интерьера на внутренней поверхности комнаты.
* Ames-img.py – позволяет сопоставить растровые изображения плоским элементам интерьера комнаты и получить их в «искаженной» перспективе комнаты Эймса. Программа поддерживает практически все растровые форматы изображений: PNG, JPG, TIFF и т.д. Предпочтительно использовать формат PNG.
* Img-projection.py – позволяет получить проекцию единственного растрового изображения на произвольно расположенную плоскость в 3D пространстве через заданную точку зрения.

Программы доступны на условиях открытой лицензии и могут быть использованы без каких-либо ограничений. Разработчик не несет ответственности за результаты использования программ третьими лицами. Тексты программ и сопутствующие материалы доступны в сети Интернет по адресу: <https://github.com/mmshihov/handmade/tree/master/eimes-room>.

## Обоснование разработки

Чтобы создать выкройку комнаты Эймса нужно выполнить много геометрических построений (см. теорию выше). Самостоятельно нарисовать линейные узоры на полу, потолке и стенах также отнимает много времени. Выполнить более-менее реалистичный рисунок, да еще и в перспективном искажении вручную для человека без профессиональных художественных навыков практически невозможно.

В то же время хочется попробовать несколько вариантов комнаты Эймса, изменяя:

* размеры комнаты;
* положение точки зрения;
* глубину искажения;
* расположение и внешний вид элементов интерьера.

Поэтому было принято решение написать несколько программ, позволяющих задать перечисленные выше характеристики и получить:

* выкройки пола, потолка и стен комнаты Эймса;
* искаженные растровые изображения внутреннего убранства комнаты (например, часов, картин, камина, дверей и т.д.).

Таким образом, ручной труд сведется к склеиванию объемного макета комнаты из распечатанных на принтере выкроек и фрагментов искаженных изображений интерьера комнаты.

# Особенности разработки

В качестве формата для выкроек комнаты был выбран формат SVG, так как это текстовый формат и изображение в этом формате может быть легко сформировано встроенными в Python функциями работы со строками.

Растровые изображения обрабатываются с помощью библиотеки Pillow. Это позволяет не думать о форматах растровых файлов и особенностях их обработки. На любой платформе библиотека Pillow может быть установлена парой команд в терминале:

**> python3 -m pip install --upgrade pip**

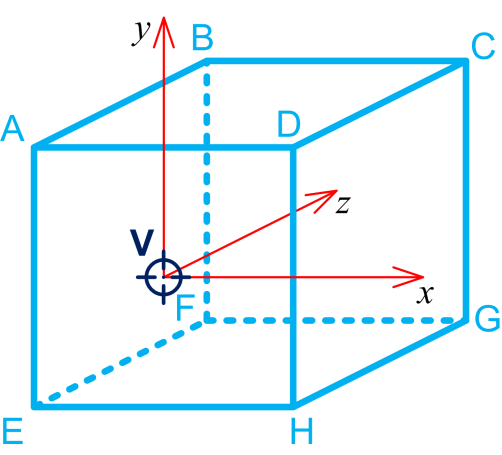
**> python3 -m pip install --upgrade Pillow**

# Краткие руководства пользователя

Все программы написаны на языке Python версии 3.11. Работают на любой платформе (операционной системе) с установленным интерпретатором Python третьей версии и выше.

## Руководство пользователя программы «ames.py»

Программа генерирует выкройки комнаты Эймса: пол, потолок и стены в векторном формате SVG.



Чтобы получить выкройки, нужно найти в тексте программы и задать значения констант для координаты ключевой точки E (см. рисунок выше) а также размеров комнаты:

**E = [-20, -20, 0]**

**WIDTH = 40 # ширина EH**

**HEIGHT = 40 # высота EA**

**DEPTH = 40 # глубина EF**

Размеры и координаты измеряются в миллиметрах. Также важный параметр --- это масштаб глубины комнаты Эймса:

**SCALE = 2**

При необходимости также можно поменять координаты точки зрения V, которая по умолчанию находится в начале координат:

**V = [0, 0, 0]**

Если вы хотите другие узоры, внимательно прочитайте комментарии в тексте программы и сделайте их самостоятельно.

Файлы сгенерированных выкроек создаются в подкаталоге 'output/svgs/\*' основного каталога программы. Если этот подкаталог отсутствует, создайте его.

## Руководство пользователя "ames-img.py"

Программа генерирует искаженные картинки узоров для выкроек комнаты Эймса, которые создаются программой "ames.py".

Настройки программы аналогичны настройкам программы "ames.py". Дополнительно вам нужно разобраться как генерируются узоры и задать пути к файлам искажаемых изображений. По умолчанию файлы примеров располагаются в подкаталоге 'output/pictures/'.

Файлы сгенерированных изображений находятся в подкаталоге 'output/pictures/\*' основного каталога программы.

## Руководство пользователя "img-projection.py"

Программа в общем случае не имеет никакого отношения к комнате Эймса и просто генерирует проекцию картинки на плоскость через заданную точку зрения. Она может быть использована для генерации нестандартных изображений для комнаты Эймса.

# Результаты работы программ

Экспериментируем с выкройками, изменяя точку зрения и глубину искажения.

Приведенный ниже рисунок – это выкройка с точкой зрения по центру комнаты и коэффициентом искажения 2.



Сдвинем точку зрения вбок влево.



Сдвинем точку зрения вбок влево и вниз.



Увеличим коэффициент искажения до 3-х:



Приведем также результаты нескольких экспериментов с растровым изображением героя мультфильма "Лило и стич" (изображение взято с сайта: <https://avatarko.ru/kartinka/32117>).

На изображения в перспективном искажении нужно смотреть снизу, с расстояния 30 - 50 см., меняя наклон изображения по вертикали. До тех пор, пока оно не покажется вам таким же, как исходное изображение, на которое вы смотрите фронтально:









Приложение А

(Исходные тексты программы «ames.py»)

# Это программа-генератор выкроек стен, потолка и пола комнаты Эймса.

# Выкройки будут генерироваться в векторном формате SVG

# Бери, печатай, вырезай, клей.

# Обозначения элементов комнаты Эймса

# возьмем как на рисунке:

# "../../../eimes-room/beamer/figs/ames-xyz.png"

#

# Обозначения углов комнаты:

# E = левый--нижний--ближний

# F = левый--нижний--дальний

# A = левый--верхний-ближний

# B = левый--верхний-дальний

# H = правый-нижний--ближний

# G = правый-нижний--дальний

# D = правый-верхний-ближний

# C = правый-верхний-дальний

#

# Т.е.:

# floor - пол (EFGH);

# ceil - потолок (ABCD);

# leftWall - левая стена (EABF);

# frontWall - передняя стена (FBCG);

# rightWall - правая стена (GCDH);

#

# Одноименные файлы выкроек вы получите на выходе.

# Трехментые координаты будут представлены списком из трех чисел.

# Двумерные (для выкроек стен, потолка и пола комнаты) --- списком из двух чисел.

# 3D точка: [x, y, z], 2D: [x, y].

# Измерения: x - ширина (право-лево), y - высота (верх-низ), z - глубина (вблизи-вдали)

# Измеряем всё в миллиметрах!

# Программа не проверяет влезет ли выкройка на A4-й лист бумаги --- подгоняйте сами.

# Точку зрения V поместим в начало координат:

V = [0, 0, 0]

# Плоскости потолка и пола параллельны плоскости xOz;

# плоскости левой и правой стен параллельны yOz;

# передняя стена параллельна xOy.

# Чтобы задать положение комнаты в 3D, зададим координаты точки

E = [-100, -100, 0]

# т.е. E --- это левый-нижний-ближний угол комнаты. А также измерения комнаты:

WIDTH = 200

HEIGHT = 200

DEPTH = 200

# Координаты остальных вершин комнаты получаются автоматически от базовой точки E:

# + WIDTH, + HEIFHT, + DEPTH

A = [E[0], E[1] + HEIGHT, E[2]]

D = [E[0] + WIDTH, E[1] + HEIGHT, E[2]]

H = [E[0] + WIDTH, E[1], E[2]]

F = [E[0], E[1], E[2] + DEPTH]

B = [E[0], E[1] + HEIGHT, E[2] + DEPTH]

C = [E[0] + WIDTH, E[1] + HEIGHT, E[2] + DEPTH]

G = [E[0] + WIDTH, E[1], E[2] + DEPTH]

# Также не проверяется видимость всей комнаты из точки V (проверяйте сами)

# На стенах, полу и потолке комнаты могут быть линейные узоры.

# На ближней стороне пола, потолка, левой и правой стен будем добавлять

# треугольную маркировку точки зрения V (чтобы не запутаться).

# Это тоже будет узор (как будто маркировка сделана прямо на стенах

# комнаты). Цвет и размеры этой маркировки:

MARK\_COLOR = "lawngreen"

MARK\_WIDTH = 8 # миллиметры

MARK\_HEIGHT = 10

# Элемент узора (например, одна кафельная плитка на полу) в этой программе

# представляет собой замкнутую фигуру из нескольких 2D или 3D точек (хранимых списком),

# лежащих в одной плоскости, залитую одним цветом:

class Pattern:

def \_\_init\_\_(self, color: str, points: list[list[float]]):

self.color = color

self.points = points

# Так как каждый элемент узора (все его точки) будет испытывать одни и те же

# линейные преобразования, то будем обрабатывать сразу массивы таких элементов.

# Определим несколько функций для генерации узоров комнаты:

# создаем пол в шахматную клеточку (выложен плиткой)

def generatePatternsChessFloor():

FIELD\_COUNT\_ON\_WIDTH = 10

COLORS = ["black", "white"]

w = (G[0] - F[0]) / FIELD\_COUNT\_ON\_WIDTH # плитка квадратная, размером w\*w

x = F[0] # начинаем генерировать клеточки с левого-нижнего-дальнего угла

z = F[2] # x, z --- координаты одного из уголоков плитки

ix = iz = 0 # номер плитки по ширине и глубине (двумерный массив плиток)

# для определения того, каким цветом красить плитку

# в шахматку ((ix+iy) mod COLORS.COUNT)

patterns = [Pattern("white", [E, F, G, H])] # первым элементом списка будет весь пол

# а только потом --- плитки:

while (x < G[0]):

z = F[2]

iz = 0

while (z > E[2]):

# может быть последние плитки будут не целыми, поэтому определяем

# ширину и глубину конкретной плитки:

wx = wz = w

if (x + wx > G[0]):

wx = G[0] - x

if (z - wz < E[2]):

wz = z - E[2]

color = COLORS[(ix + iz) % len(COLORS)] # определяем цвет добавляемой плитки

# добавляем плитку в масссив узоров

patterns.append(Pattern( # создаем узор плитки, указывая:

color, # цвет плитки

[ # координаты точек плитки, лежащей на полу (плоскость xOz):

[x, E[1], z],

[x + wx, E[1], z],

[x + wx, E[1], z - wz],

[x, E[1], z - wz]

]

))

z = z - w

iz = iz + 1

x = x + w

ix = ix + 1

# Добавим треугольную маркировку центра точки зрения

patterns.append(Pattern(

MARK\_COLOR,

[

[V[0] - MARK\_WIDTH/2, E[1], E[2] + MARK\_HEIGHT],

[V[0] + MARK\_WIDTH/2, E[1], E[2] + MARK\_HEIGHT],

[V[0], E[1], E[2]]

]

))

return patterns

# На потолке будет пять квадратных светильников (четыре по краям, один большой в центре)

def generatePatternsCeil():

LIGHT\_DIMENSION\_PER\_WIDTH = 10 # сколько светильников помещается по ширине (минимум)

COLOR = "blue"

wx = (C[0] - B[0]) / LIGHT\_DIMENSION\_PER\_WIDTH # размер светильника по ширине

wz = (B[2] - A[2]) / LIGHT\_DIMENSION\_PER\_WIDTH # размер светильника по глубине

w = wx

if (wz < w): w = wz

patterns = [Pattern("white", [A, B, C, D])] # первым элементом списка будет весь потолок

# светильники:

x = B[0] + w; z = B[2] - 2\*w # левый-дальний

patterns.append(Pattern(COLOR,[ [x, A[1], z], [x + w, A[1], z], [x + w, A[1], z + w], [x, A[1], z + w]]))

x = B[0] + w; z = A[2] + w # левый-ближний

patterns.append(Pattern(COLOR,[ [x, A[1], z], [x + w, A[1], z], [x + w, A[1], z + w], [x, A[1], z + w]]))

x = C[0] - 2\*w; z = C[2] - 2\*w # правый-дальний

patterns.append(Pattern(COLOR,[ [x, A[1], z], [x + w, A[1], z], [x + w, A[1], z + w], [x, A[1], z + w]]))

x = C[0] - 2\*w; z = D[2] + w # правый-ближний

patterns.append(Pattern(COLOR,[ [x, A[1], z], [x + w, A[1], z], [x + w, A[1], z + w], [x, A[1], z + w]]))

# центральный (больше остальных в два раза линейно)

x = (B[0] + C[0]) / 2; z = (A[2] + B[2]) / 2

patterns.append(Pattern(

COLOR,

[

[x - w, A[1], z - w],

[x - w, A[1], z + w],

[x + w, A[1], z + w],

[x + w, A[1], z - w],

]))

# Добавим треугольную маркировку центра точки зрения

patterns.append(Pattern(

MARK\_COLOR,

[

[V[0] - MARK\_WIDTH/2, A[1], A[2] + MARK\_HEIGHT],

[V[0] + MARK\_WIDTH/2, A[1], A[2] + MARK\_HEIGHT],

[V[0], A[1], A[2]]

]

))

return patterns

# На всех стенах есть фартук:

BORDER\_HEIGHT = HEIGHT \* 3 / 8

BORDER\_COLOR = "green"

# левая стена с дверью

def generatePatternsLeftWall():

DOOR\_HEIGHT = HEIGHT \* 4 / 5

DOOR\_WIDTH = DOOR\_HEIGHT / 3

DOOR\_COLOR = "grey"

patterns = [Pattern("white", [E, A, B, F])] # стена целиком

# фартук

patterns.append(Pattern(

BORDER\_COLOR,

[

E,

[E[0], E[1] + BORDER\_HEIGHT, E[2]],

[F[0], F[1] + BORDER\_HEIGHT, F[2]],

F

]))

# дверь

patterns.append(Pattern(

DOOR\_COLOR,

[

[F[0], F[1], F[2] - DOOR\_WIDTH/2],

[F[0], F[1], F[2] - DOOR\_WIDTH/2 - DOOR\_WIDTH],

[F[0], F[1] + DOOR\_HEIGHT, F[2] - DOOR\_WIDTH/2 - DOOR\_WIDTH],

[F[0], F[1] + DOOR\_HEIGHT, F[2] - DOOR\_WIDTH/2]

]))

# Добавим треугольную маркировку центра точки зрения

patterns.append(Pattern(

MARK\_COLOR,

[

[E[0], V[1] - MARK\_WIDTH/2, E[2] + MARK\_HEIGHT],

[E[0], V[1] + MARK\_WIDTH/2, E[2] + MARK\_HEIGHT],

[E[0], V[1], E[2]]

]

))

return patterns

# правая стена с такой же дверью и картиной

def generatePatternsRightWall():

DOOR\_HEIGHT = HEIGHT \* 4 / 5

DOOR\_WIDTH = DOOR\_HEIGHT / 3

DOOR\_COLOR = "grey"

PICTURE\_HEIGHT = HEIGHT / 4

PICTURE\_WIDTH = DEPTH / 8

PICTURE\_COLOR = "blue"

patterns = [Pattern("white", [H, D, C, G])] # стена целиком

# фартук

patterns.append(Pattern(

BORDER\_COLOR,

[

H,

[H[0], H[1] + BORDER\_HEIGHT, H[2]],

[G[0], G[1] + BORDER\_HEIGHT, G[2]],

G

]))

# дверь

patterns.append(Pattern(

DOOR\_COLOR,

[

[G[0], G[1], G[2] - DOOR\_WIDTH/2],

[G[0], G[1], G[2] - DOOR\_WIDTH/2 - DOOR\_WIDTH],

[G[0], G[1] + DOOR\_HEIGHT, G[2] - DOOR\_WIDTH/2 - DOOR\_WIDTH],

[G[0], G[1] + DOOR\_HEIGHT, G[2] - DOOR\_WIDTH/2]

]))

# картина по центру стены

y = (C[1] + G[1])/2

z = (H[2] + G[2])/2

patterns.append(Pattern(

PICTURE\_COLOR,

[

[G[0], y - PICTURE\_HEIGHT/2, z - PICTURE\_WIDTH/2],

[G[0], y - PICTURE\_HEIGHT/2, z + PICTURE\_WIDTH/2],

[G[0], y + PICTURE\_HEIGHT/2, z + PICTURE\_WIDTH/2],

[G[0], y + PICTURE\_HEIGHT/2, z - PICTURE\_WIDTH/2]

]))

# Добавим треугольную маркировку центра точки зрения

patterns.append(Pattern(

MARK\_COLOR,

[

[H[0], V[1] - MARK\_WIDTH/2, H[2] + MARK\_HEIGHT],

[H[0], V[1] + MARK\_WIDTH/2, H[2] + MARK\_HEIGHT],

[H[0], V[1], H[2]]

]

))

return patterns

# дальняя стена с камином или картиной по центру

def generatePatternsFrontWall():

PICTURE\_HEIGHT = HEIGHT / 2

PICTURE\_WIDTH = WIDTH / 2

PICTURE\_COLOR = "blue"

patterns = [Pattern("white", [B, C, G, F])] # стена целиком

# фартук

patterns.append(Pattern(

BORDER\_COLOR,

[

F,

[F[0], F[1] + BORDER\_HEIGHT, F[2]],

[G[0], G[1] + BORDER\_HEIGHT, G[2]],

G

]))

# картина по центру стены

x = (F[0] + G[0])/2

y = (F[1] + B[1])/2

patterns.append(Pattern(

PICTURE\_COLOR,

[

[x - PICTURE\_WIDTH/2, y - PICTURE\_HEIGHT/2, F[2]],

[x + PICTURE\_WIDTH/2, y - PICTURE\_HEIGHT/2, F[2]],

[x + PICTURE\_WIDTH/2, y + PICTURE\_HEIGHT/2, F[2]],

[x - PICTURE\_WIDTH/2, y + PICTURE\_HEIGHT/2, F[2]]

]))

return patterns

# сохраняем узоры в соответствующих переменных:

floorPatterns = generatePatternsChessFloor()

ceilPatterns = generatePatternsCeil()

leftWallPatterns = generatePatternsLeftWall()

rightWallPatterns = generatePatternsRightWall()

frontWallPatterns = generatePatternsFrontWall()

# функция получения вектора из двух точек

def vector3D(a, b):

return [(b[0] - a[0]), (b[1] - a[1]), (b[2] - a[2])]

# Определим функцию, выполняющую скалярное произведение

# векторов a, b в 3D:

def scalarMul3D(a, b):

# по учебнику из раздела аналитической геометрии

# a = [x1,y1,z1]

# b = [x2,y2,z2]

(x1, y1, z1) = (a[0], a[1], a[2])

(x2, y2, z2) = (b[0], b[1], b[2])

return x1\*x2 + y1\*y2 + z1\*z2

# Определим функцию, выполняющую векторное произведение

# векторов a, b в 3D:

def vectorMul3D(a, b):

# по учебнику из раздела аналитической геометрии

# a = [x1,y1,z1]

# b = [x2,y2,z2]

(x1, y1, z1) = (a[0], a[1], a[2])

(x2, y2, z2) = (b[0], b[1], b[2])

return [y1\*z2 - z1\*y2, -x1\*z2 + z1\*x2, x1\*y2 - y1\*x2]

# Введем класс для работы с прямой в 3D:

# x = P0 + V0\*t

# y = P1 + V1\*t

# z = P2 + V2\*t

# --- т.е. прямая задается начальной координатой P и вектором направления V

class Line3D:

def \_\_init\_\_(self, a, b): # конструируется (уравнение Ax+By+Cz+D=0) по трем точкам

self.V = vector3D(a, b)

self.P = [a[0], a[1], a[2]]

def point(self, t):

return [

self.P[0] + self.V[0]\*t,

self.P[1] + self.V[1]\*t,

self.P[2] + self.V[2]\*t

]

# Введем класс для работы с плоскостью (все, что нам нужно от

# полскости --- это находить точку, в которой она пересекается с 3D прямой)

class Plane3D:

def \_\_init\_\_(self, a, b, c): # конструируется (уравнение Ax+By+Cz+D=0) по трем точкам

abVector = vector3D(a, b)

acVector = vector3D(a, c)

self.ABC = vectorMul3D(abVector, acVector) # вектор нормали плоскости

self.D = -(self.ABC[0]\*a[0] + self.ABC[1]\*a[1] + self.ABC[2]\*a[2])

# находит точку пересечения себя с прямой l

def intersectionPointWithLine(self, l:Line3D):

scalar = scalarMul3D(self.ABC, l.V)

if (scalar == 0):

raise ValueError("Прямая и плоскость не пересекаются!")

t = -(scalarMul3D(self.ABC, l.P) + self.D)/scalar

return l.point(t)

# Определим новые точки C и G (amesC, amesG), отодвинув их на луче зрения.

# Напомним, что луч зрения исходит из точки V.

# Масштаб искажения:

SCALE = 2

amesC = Line3D(V, C).point(SCALE)

amesG = Line3D(V, G).point(SCALE)

# По построению точки B,D,F,H остаются неизменными:

amesB = B

amesD = D

amesF = F

amesH = H

# Теперь можно определить плоскости пола, потолка, правой и дальней стены

# комнаты Эймса:

amesFloor = Plane3D(amesF, amesH, amesG)

amesCeil = Plane3D(amesB, amesD, amesC)

amesRightWall = Plane3D(amesD, amesH, amesC)

amesFrontWall = Plane3D(amesB, amesF, amesC)

# Осталось найти плоскость левой стены. Луч зрения VA пересечет потолок

# в точке A', которая также принадлежит левой стене

amesA = amesCeil.intersectionPointWithLine(Line3D(V,A))

# Таким образом, плоскость левой стены тоже определена:

amesLeftWall = Plane3D(amesA, amesB, amesF)

# Определим функцию, которая строит проекцию точки, на которую смотрят

# из точки V на заданную плоскость:

def viewProjectionPointOnPlane(point, plane:Plane3D):

return plane.intersectionPointWithLine(Line3D(V, point))

# Определим также функцию, которая проецирует целый массив узоров

def patternsProjection(patterns:list[Pattern], plane:Plane3D):

projections = []

for pattern in patterns:

points = []

for point in pattern.points:

points.append(viewProjectionPointOnPlane(point, plane)) # спроецировали каждую точку

projections.append(Pattern(pattern.color, points))

return projections

# Находим проекции узоров

amesFloorPatterns = patternsProjection(floorPatterns, amesFloor)

amesCeilPatterns = patternsProjection(ceilPatterns, amesCeil)

amesLeftWallPatterns = patternsProjection(leftWallPatterns, amesLeftWall)

amesRightWallPatterns = patternsProjection(rightWallPatterns, amesRightWall)

amesFrontWallPatterns = patternsProjection(frontWallPatterns, amesFrontWall)

# Теперь проекции узоров нужно повернуть так, чтобы они "легли" например, на

# плоскость xOy (все узоры лежат в одной плоскости по определению).

# Делать это можно с помощью матрицы поворотов. Поворачивать придется

# дважды:

# \* сначала вокруг оси Oz, на угол между ортом Ox и линией пересечения

# плоскости с xOy

# \* затем, после предыдущего поворота, вокруг оси Ox, на угол между ортом Oy

# и линией пересечения плоскости с yOz

# После этого, координаты z должны стать одинаковыми

# Определим функцию умножения вектора на матрицу:

def vectorMulMatrix3D(v, m):

r = [0,0,0]

for i in range(3):

for j in range(3):

r[i] += v[j] \* m[j][i]

return r

# Определим функцию умножения матрицу на матрицу

def matrixMulMatrix3D(m1,m2):

r = [[0,0,0],

[0,0,0],

[0,0,0]]

for i in range(3):

for j in range(3):

for k in range(3):

r[i][j] += m1[i][k]\*m2[k][j]

return r

# достанем функцию квадратного корня из пакета math

from math import sqrt;

# Определим функию поворота произвольно расположенной плоскости

# так, чтобы она стала параллельной плоскости xOy.

# Поворот сохраняет расстояние между точками, соответственно

# такое преобразование позволяет получить 2D выкройку на плоскости.

# Выполняется два поворота: первый --- вокруг оси Oz,

# второй --- вокруг Oy

# ---

# Функция возвращает результирующую матрицу, на которую можно

# умножать любую 3D точку, лежащую в исходной плоскости и таким образом

# получать ее координаты на плоскости

def matrixFor2D\_xOy(plane:Plane3D):

# находим синус и косинус угла поворота вектора нормали плоскости вокруг оси

# Oz (после поворота компонента y (plane.ABC[1]) вектора нормали должна стать

# нулевой)

s = plane.ABC[1]/sqrt(plane.ABC[0]\*plane.ABC[0] + plane.ABC[1]\*plane.ABC[1])

c = plane.ABC[0]/sqrt(plane.ABC[0]\*plane.ABC[0] + plane.ABC[1]\*plane.ABC[1])

rm1 = [ # так выглядит матрица поворота вокруг Oz

[ c,-s, 0],

[ s, c, 0],

[ 0, 0, 1]

]

# повернули исходный вектор нормали (y - составляющая == 0)

newABC = vectorMulMatrix3D(plane.ABC, rm1)

# синус и косинус угла поворота вокруг Oy

s = newABC[0]/sqrt(newABC[0]\*newABC[0] + newABC[2]\*newABC[2])

c = newABC[2]/sqrt(newABC[0]\*newABC[0] + newABC[2]\*newABC[2])

rm2 = [ # матрица поворота вокруг Oy

[ c, 0, s],

[ 0, 1, 0],

[-s, 0, c]

]

return matrixMulMatrix3D(rm1, rm2) # два поворота в одной матрице!!!

# (v\*rm1)\*rm2 == v\*(rm1\*rm2)

# Определим функцию, которая преобразует точки массива узоров,

# умножая их на матрицу m (с помощью матриц можно, например,

# вращать и отражать точки(вектора))

def patternsMulMatrix3D(patterns:list[Pattern], m):

newPatterns = []

for pattern in patterns:

points = []

for point in pattern.points:

points.append(vectorMulMatrix3D(point, m)) # умножаем каждую точку

newPatterns.append(Pattern(pattern.color, points))

return newPatterns

# Определим функцию, которая поворачивает массив узоров, которые лежат в

# одной плоскости (аргумент plane)

def patternsRotationFor2D\_xOy(patterns:list[Pattern], plane:Plane3D):

return patternsMulMatrix3D(patterns, matrixFor2D\_xOy(plane))

# получили выкройки на плоскости (координаты z у всех точек выкройки для каждого

# узора в списке --- одинаковые)

ames2D\_xOy\_FloorPatterns = patternsRotationFor2D\_xOy(amesFloorPatterns, amesFloor)

ames2D\_xOy\_CeilPatterns = patternsRotationFor2D\_xOy(amesCeilPatterns, amesCeil)

ames2D\_xOy\_LeftWallPatterns = patternsRotationFor2D\_xOy(amesLeftWallPatterns, amesLeftWall)

ames2D\_xOy\_RightWallPatterns = patternsRotationFor2D\_xOy(amesRightWallPatterns, amesRightWall)

ames2D\_xOy\_FrontWallPatterns = patternsRotationFor2D\_xOy(amesFrontWallPatterns, amesFrontWall)

# сделаем также выкройки базовой комнаты (все, кроме дальней стены, которая и

# xOy, нужно будет повернуть):

room2D\_xOy\_FloorPatterns = patternsRotationFor2D\_xOy(floorPatterns, Plane3D(E, F, G))

room2D\_xOy\_CeilPatterns = patternsRotationFor2D\_xOy(ceilPatterns, Plane3D(A, B, D))

room2D\_xOy\_LeftWallPatterns = patternsRotationFor2D\_xOy(leftWallPatterns, Plane3D(E, A, F))

room2D\_xOy\_RightWallPatterns = patternsRotationFor2D\_xOy(rightWallPatterns, Plane3D(H, D, G))

room2D\_xOy\_FrontWallPatterns = frontWallPatterns

# Определим функцию, возвращающую параметры прямогугольной области

# (xMin, yMin, width, height), которая содержит все узоры в заданном массиве узоров

# Функция работает с выкройкой (т.е. использует только координаты x,y трехмерной точки

# [x,y,z])

def patternsArea2D\_xOy(patterns: list[Pattern]):

if (len(patterns) == 0):

raise ValueError("Пустые массивы узоров не принимаются!")

xMin = xMax = patterns[0].points[0][0]

yMin = yMax = patterns[0].points[0][1]

for pattern in patterns:

for point in pattern.points:

x = point[0]

if x < xMin: xMin = x

if x > xMax: xMax = x

y = point[1]

if y < yMin: yMin = y

if y > yMax: yMax = y

return (xMin, yMin, xMax - xMin, yMax - yMin)

# Формируем SVG-файл. Он текстовый, его можно читать человеку и даже понимать,

# что там нарисовано. Определим вспомогательные функции для формирования

# текстовых элементов SVG файла.

# Все наши измерения в миллиметрах, к счастью SVG работает с милиметрами

# (и не только). Но по умолчанию он работает с пикселами. Т.е., если для

# координаты или размерности указано число, то это будет трактоваться

# как количество пикселей. Чтобы работать в миллиметрах, нужно указывать

# постфикс "mm". Например, "height=100mm". Напишем функцию, форматирующую

# наше число в правильную SVG-шную строку

def svgFormatInMm(digit):

return "{:.2f}mm".format(digit)

def svgFormatInViewPort(digit):

return "{:.2f}".format(digit)

# Открывающий тэг заголовка SVG-файла "<svg ...>""

# Имеет парную закрывающую часть "</svg>"

# Между открывающей и закрывающей частями будут находится команды рисования

def SVG\_HEAD(width, height, viewBox):

# в питоне вот так можно в тройных кавычках писать многострочный текст:

return '''\

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<svg version="1.1"

width="{0}"

height="{1}"

viewBox="{2}"

preserveAspectRatio="none"

xmlns="http://www.w3.org/2000/svg">

'''.format(width, height, viewBox)

def SVG\_TAIL():

return '</svg>'

def SVG\_POLYGON\_FROM\_PATTERN(pattern:Pattern):

pointsStr = ""

for point in pattern.points:

x, y = point[0], point[1]

if len(pointsStr) == 0:

pointsStr = pointsStr + '{0},{1}'.format(svgFormatInViewPort(x), svgFormatInViewPort(y))

else:

pointsStr = pointsStr + ' {0},{1}'.format(svgFormatInViewPort(x), svgFormatInViewPort(y))

return '<polygon points="{0}" fill="{1}"/>'.format(pointsStr, pattern.color)

def SVG\_POLYLINE\_FROM\_PATTERN(pattern:Pattern):

pointsStr = ""

for point in pattern.points:

x, y = point[0], point[1]

if len(pointsStr) == 0:

pointsStr = pointsStr + '{0},{1}'.format(svgFormatInViewPort(x), svgFormatInViewPort(y))

else:

pointsStr = pointsStr + ' {0},{1}'.format(svgFormatInViewPort(x), svgFormatInViewPort(y))

# добавляем последнюю точку, чтобы провести замкнутую линию

point = pattern.points[0]

x, y = point[0], point[1]

pointsStr = pointsStr + ' {0},{1}'.format(svgFormatInViewPort(x), svgFormatInViewPort(y))

return '<polyline points="{0}" stroke="black" stroke-width="1px" stroke-opacity="0.3" fill="none"/>'.format(pointsStr)

def SVG\_FOR\_PATTERNS(patterns:list[Pattern]):

xMin, yMin, w, h = patternsArea2D\_xOy(patterns)

viewBox = '{} {} {} {}'.format(svgFormatInViewPort(xMin), svgFormatInViewPort(yMin), svgFormatInViewPort(w), svgFormatInViewPort(h))

str = SVG\_HEAD(svgFormatInMm(w), svgFormatInMm(h), viewBox)

for pattern in patterns:

str = str + "\n" + SVG\_POLYGON\_FROM\_PATTERN(pattern)

str = str + "\n" + SVG\_POLYLINE\_FROM\_PATTERN(patterns[0])

str = str + SVG\_TAIL()

return str

# Сохраняем массив 2D узоров файл (в том же каталоге, что и скрипт)

def savePatternsToSvg(fileName:str, patterns:list[Pattern]):

file = open('{}.svg'.format(fileName), mode='wt', encoding='utf-8')

file.write(SVG\_FOR\_PATTERNS(patterns))

file.close()

# Так как мы формировали узор в комнате, как бы смотря "изнутри", то

# получается, что на проекции (которая по вектору одного направления)

# мы как бы смотрим на узор стены "снаружи" комнаты).

# Поэтому часть узоров нужно печатать в отражении.

# Поэтому определим несколько зеркально "отражащюих" функций:

# отражение по xOz (y = -y)

def mirrorOx(patterns: list[Pattern]):

m = [ [1, 0, 0], # это матрица отражения y = -y

[0,-1, 0],

[0, 0, 1] ]

return patternsMulMatrix3D(patterns, m)

# отражение по Oy (x = -x)

def mirrorOy(patterns: list[Pattern]):

m = [ [-1, 0, 0],

[ 0, 1, 0],

[ 0, 0, 1] ]

return patternsMulMatrix3D(patterns, m)

# Сохраняем узоры в SVG (ищите файлы в папке "output", там же, где лежит скрипт).

# Часть узоров делаем в отражении (научный тык).

savePatternsToSvg('output/svgs/ames\_Floor', mirrorOx(ames2D\_xOy\_FloorPatterns))

savePatternsToSvg('output/svgs/ames\_Ceil', ames2D\_xOy\_CeilPatterns)

savePatternsToSvg('output/svgs/ames\_LeftWall', ames2D\_xOy\_LeftWallPatterns)

savePatternsToSvg('output/svgs/ames\_RightWall', ames2D\_xOy\_RightWallPatterns)

savePatternsToSvg('output/svgs/ames\_FrontWall', mirrorOy(ames2D\_xOy\_FrontWallPatterns))

savePatternsToSvg('output/svgs/base\_Floor', room2D\_xOy\_FloorPatterns)

savePatternsToSvg('output/svgs/base\_Ceil', room2D\_xOy\_CeilPatterns)

savePatternsToSvg('output/svgs/base\_LeftWall', room2D\_xOy\_LeftWallPatterns)

savePatternsToSvg('output/svgs/base\_RightWall', mirrorOx(room2D\_xOy\_RightWallPatterns))

savePatternsToSvg('output/svgs/base\_FrontWall', room2D\_xOy\_FrontWallPatterns)

print("Done. Use \*.svg files in the 'output/svgs/\*' directory")