ТЗ на корректирование координат точки
(Wall Correction)

В CPP данный функционал реализован в файле:

“**NavigatorLib/src/Navigator/Accel/TrajectoryDetection.cpp** ”

Метод “*Math::Position3D* ***TrajectoryDetection::step*** *(double dx, double dy, bool isStationary*)”.

В методе есть такое место:

```
if (rMesh2 != nullptr) {
        // Check map edges
        if (config.meshConfig.useMapEdges) {
                posX = rMesh2->checkX(posX);    // Put into map boundaries
                posY = rMesh2->checkY(posY);
        }
        // Check walls
        if (config.meshConfig.useWalls) {
                if (rMesh2->checkWall(posX0, posY0, posX, posY)) {
                        if (!rMesh2->checkWall(posX0, posY0, posX0, posY))
                                posX = posX0;
                        else if (!rMesh2->checkWall(posX0, posY0, posX, posY0))
                                posY = posY0;
                }
        }
}
else
        delta = {0, 0};

// Mesh + mask correction, post process uses the coarse mesh rMesh !
Math::Position3D position(posX, posY, 0.0);
if (rMesh)
        position = rMesh->process(position, config.meshConfig);
return position;
```

Что предлагаю и что нужно реализовать, так как коррекция по стенам на данный момент в ряде случаев не срабатывает.

<u>Предлагаю:</u>

1. Если включена коррекция по стенам, то коррекцию по карте по Mesh2 делать в любом случае, независимо от того, включена или нет коррекция по карте. Причина - преобразование произвольной точки за пределами карты в узел мэша приведет к непредсказуемым результатам (хотя методы “x2ix” и “y2iy” и сработают корректно). Т.е. код:

```
posX = rMesh2->checkX(posX);    // Put into map boundaries
posY = rMesh2->checkY(posY);
```

перенести в блок “if (config.meshConfig.useWalls) { … }”.

2. Если сработало одно из условий:

```
if (!rMesh2->checkWall(posX0, posY0, posX0, posY))
        posX = posX0;
else if (!rMesh2->checkWall(posX0, posY0, posX, posY0))
        posY = posY0;
```

то после коррекции точки сразу сделать коррекцию по мэш2 вместо этого кода:

```
// Mesh + mask correction, post process uses the coarse mesh rMesh !
Math::Position3D position(posX, posY, 0.0);
if (rMesh)
        position = rMesh->process(position, config.meshConfig);
return position;
```

Причина в том, что условия могут сработать, например, на границе карты. Конечная скорректированная точка может оказаться между запрещенными узлами мэша2, т.е. на границе карты. Коррекция по мэш1 может сильно ее сместить из-за разных шагов мэш1 и мэш2. Потому лучше скорректировать по мэш2.

Что нужно сделать:

3. Если оба условия

```
if (!rMesh2->checkWall(posX0, posY0, posX0, posY))
        posX = posX0;
else if (!rMesh2->checkWall(posX0, posY0, posX, posY0))
        posY = posY0;
```

не сработали, что может быть, когда точка P0=(X0; Y0) находится в углу помещения, то на данный момент следующая точка P=(X; Y) остается с неизменными координатами. Потому надо сделать:

```
else {
        posX = posX0;
        posY = posY0;
}
```

т.е. оставить координаты точки пользователя неизменными. Коррекцию по мэш2 в этом случае естественно делать не нужно.

4. Изменить функцию проверки пересечения стены, когда есть две точки, начальная и конечная (с координатами в метрах). На данный в СДК реализована функция “*RectanMesh::checkWall(double x1, double y1, double x2,*

double y2)". Общее описание алгоритма checkWall (или в MatLAB назвали этот метод fWallCorrector), на который необходимо поменять данную функцию, показано на рис. 1.

```
// Function checks whether the next point position crosses the wall on the map or not.
// This is method's version without conversion to pixels.
// INPUT PARAMETERS:
// P1          - struct, 1x2, previous particle position, in meters, (X,Y)
// P2          - struct, 1x2, next particle position, in meters, (X,Y)
// mesh        - struct, 1x2, values of mesh steps along two axes, in meters (X, Y)
// rowsN       - number of rows in masktable.out
// maskTable   - vector, Nx1, contains indecies for transformation of
//               mesh vertices according to the map mask
// OUTPUT PARAMETER:
// answer      - bool, the answer on the question in the method's header "fIsWallsCrossing()" (true = yes).
```

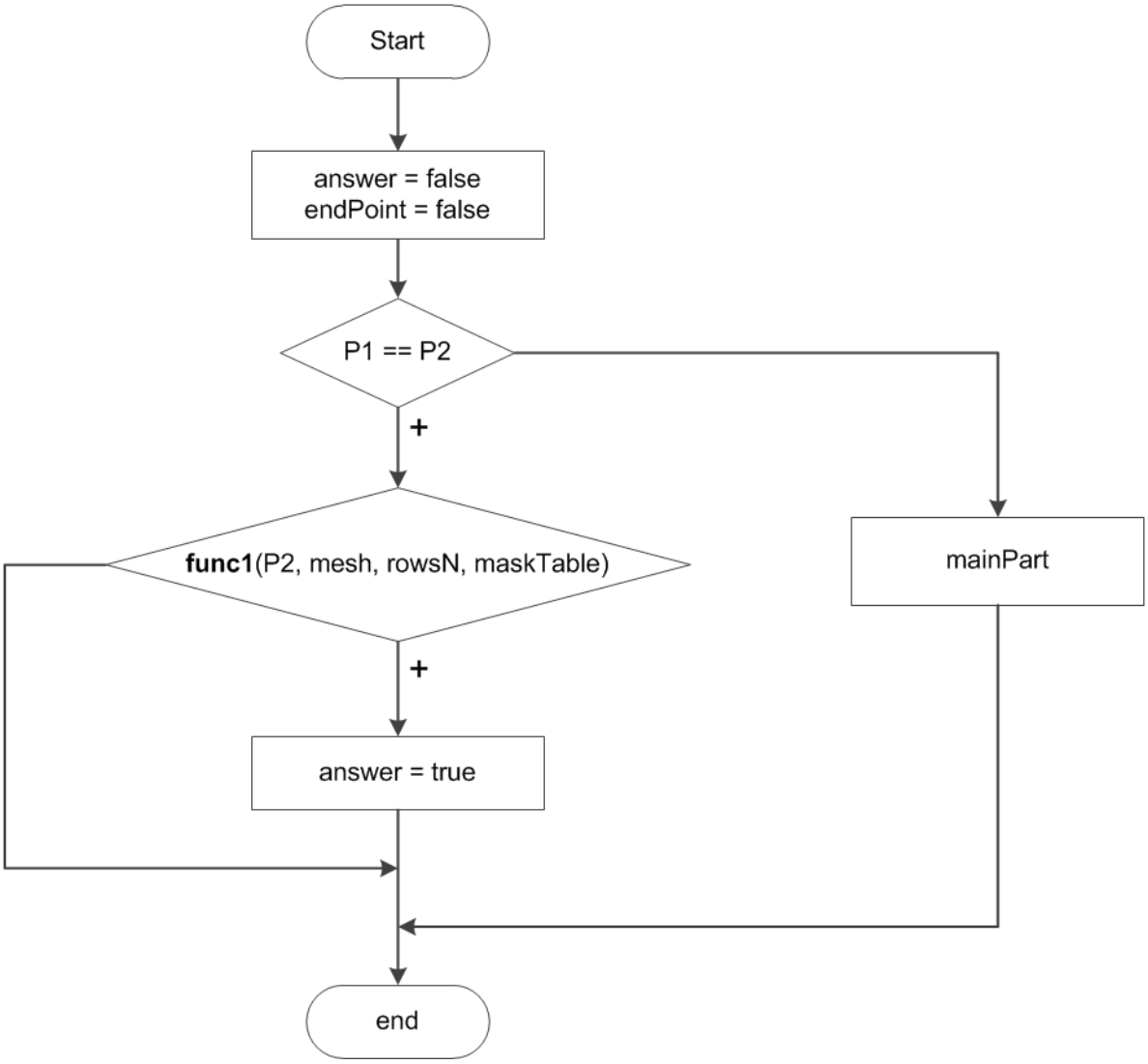

Рис. 1

Блок-схема реализации функции **func1** (по сути, она проверяет, попадает ли заданная точка на черную область карты в соответствии с заданной мэш-таблицей), показана на рис. 2.

func1 = fIsInTheBlackRegion(...)

```
// Function checks whether the particle is on the black region of the map or not
// This is method's version without conversion to pixels.
// INPUT PARAMETERS:
// rawCoords  - struct, 1x2, raw values of estimated coordinates, in meters, (X,Y)
// mesh       - vector, 1x2, values of mesh steps along two axes, in meters (X,Y)
// rowsN      - number of rows in masktable.out
// maskTable  - vector, Nx1, contains indecies for transformation of
//                mesh vertices according to the map mask
// OUTPUT PARAMETER:
// answer    - bool, the answer on the question in the method's header (true=yes)
```

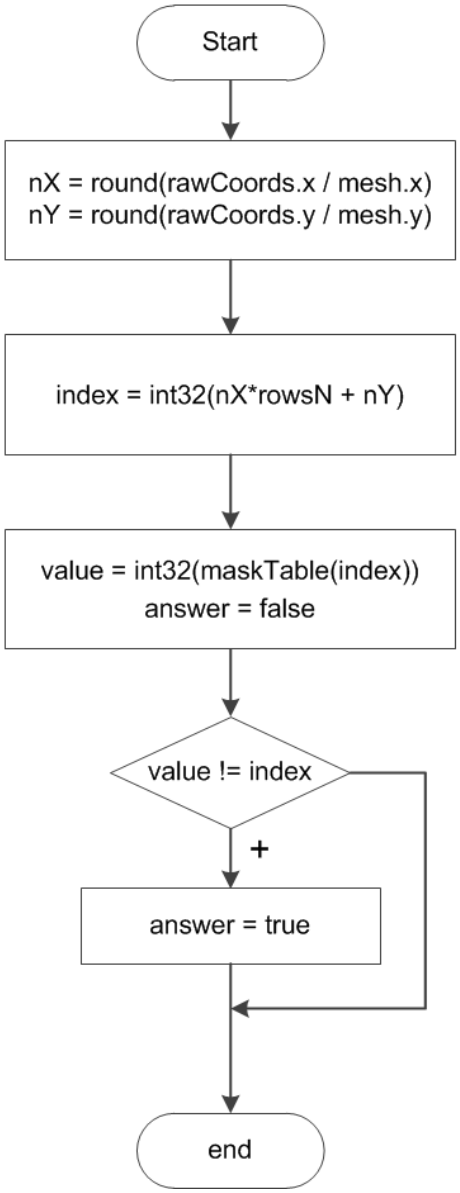

Рис. 2

Наконец основная часть алгоритма приведена на рис. 3. Версию алгоритма на рис. 3 в большом формате можно скачать по ссылке - https://www.dropbox.com/s/o3uckpc3261kngq/WallCorrection%233.png?dl=0

Общая идея алгоритма проверки пересечения стен

Рис. 4 показывает общую схему проверки. Пусть алгоритм Навигатора говорит, что следующей точкой должна быть точка P2. При этом предполагаем, что точка P2 попадает в границы карты (если не попадает, то сначала принудительно делается коррекция по границам карты при помощи meshTable2). Далее метод коррекции по стенам определяет, что между точками P1 и P2 есть стена. Тогда в качестве коррекции алгоритм сначала формирует точку P2' с координатой X от точки P1 и координатой Y от точки P2. Проверяет линию между P1 и P2' на наличие стены. Если стена в этом случае не обнаружена, то координаты следующей точки становятся равными P2'. Если стена обнаружена снова, тогда формируется точка P2'' с координатой X от точки P2 и координатой Y от точки P1. Алгоритм проверяет линию между P1 и P2'' на наличие стены. Если перемещение возможно, то координаты следующей точки становятся равными P2''. Если стена обнаружена, то координаты следующей точки приравниваются координатам предыдущей, т.е. P1, и в этом случае маркер пользователя на экране остается на месте.

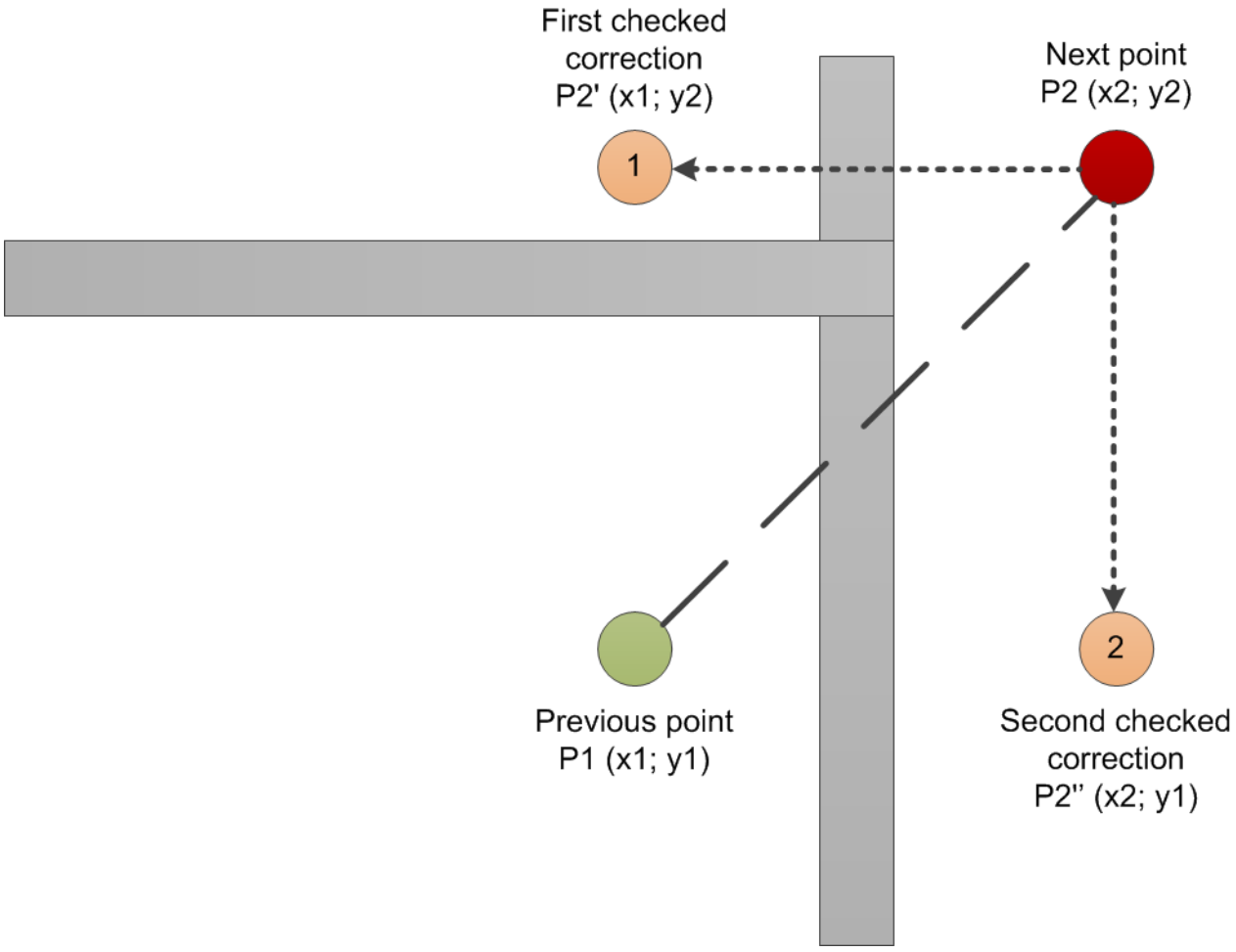

Рис. 4

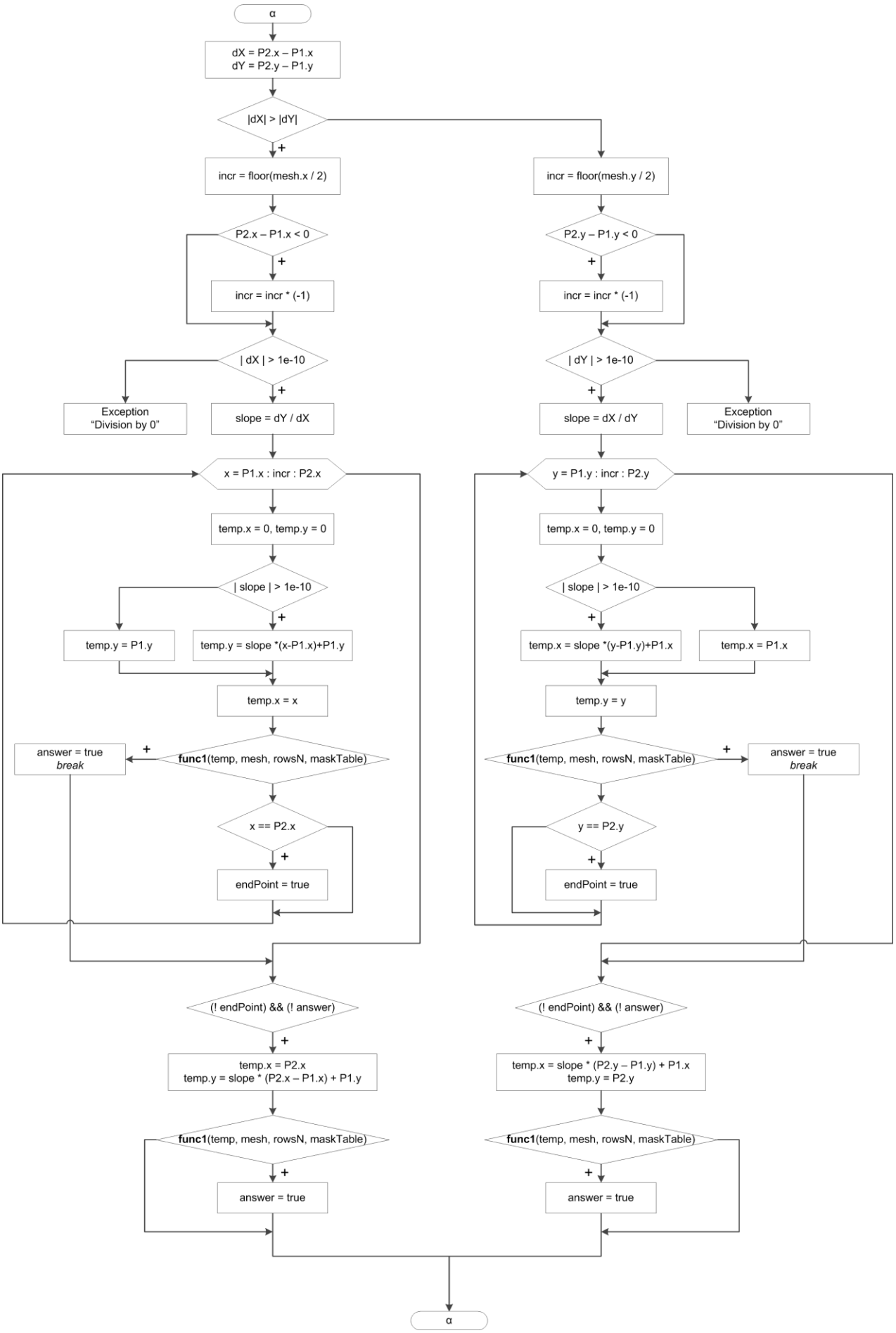
α
dX = P2.x – P1.x
dY = P2.y – P1.y
|dX| > |dY|
+
incr = floor(mesh.x / 2)
P2.x – P1.x < 0
+
incr = incr * (-1)
| dX | > 1e-10
+
Exception
“Division by 0”
slope = dY / dX
x = P1.x : incr : P2.x
temp.x = 0, temp.y = 0
| slope | > 1e-10
+
temp.y = P1.y
temp.y = slope *(x-P1.x)+P1.y
temp.x = x
answer = true
break
+
func1(temp, mesh, rowsN, maskTable)
x == P2.x
+
endPoint = true
(! endPoint) && (! answer)
+
temp.x = P2.x
temp.y = slope * (P2.x – P1.x) + P1.y
func1(temp, mesh, rowsN, maskTable)
+
answer = true
incr = floor(mesh.y / 2)
P2.y – P1.y < 0
+
incr = incr * (-1)
| dY | > 1e-10
+
Exception
“Division by 0”
slope = dX / dY
y = P1.y : incr : P2.y
temp.x = 0, temp.y = 0
| slope | > 1e-10
+
temp.x = slope *(y-P1.y)+P1.x
temp.x = P1.x
temp.y = y
func1(temp, mesh, rowsN, maskTable)
+
answer = true
break
y == P2.y
+
endPoint = true
(! endPoint) && (! answer)
+
temp.x = slope * (P2.y – P1.y) + P1.x
temp.y = P2.y
func1(temp, mesh, rowsN, maskTable)
+
answer = true
α

Рис. 3