Room Reconstruction From Meshes Created Through 3D Scanning

Zoltán Majoros

zoltan@arcanelab.com

2017

July

Abstract

This paper presents algorithms and reasoning behind my solution to reconstructing indoor spaces from 3D scanner-generated meshes. The aim is to deduce the indoor topology and overlay CAD drawings on detected walls in real-time AR passthrough view. Using Monte-Carlo sampling, a statistical analysis identifies potential wall candidates in the scanned mesh. The process involves detecting large flat surfaces and calculating their intersections to denote walls, ceilings, and floors. By employing a Monte-Carlo sampling process, the noise originating from the inherent imperfections of spatial scanning is effectively distributed randomly among the samples, thereby reducing its influence on the final results. Challenges and strategies in handling plane intersections and corner points in varying room configurations are also discussed.

1 Introduction

We are developing an application that utilizes a dedicated 3D scanner hardware attached to a mobile device. This application is capable of scanning indoor spaces, identifying the structure of the room, and displaying user-configurable CAD drawings on the walls in real-time.

2 Basic Assumptions & General Strategy

To fit a CAD drawing to a 3D scan of a room, we first need to determine the structure of the room. One approach is to detect the walls, floor, and ceiling, examine their intersections, and infer the topology of the room based on this information. However, accommodating different room topologies can increase the complexity of the detection algorithm. To address this, we introduce several assumptions about the mesh of the 3D scan:

1. Larger flat surfaces are likely to correspond to walls, floors, or ceilings.

2. The mesh is oriented such that the floor is parallel to the xz plane, with the y-axis perpendicular to the floor. This orientation is based on the gravitational force detected by the accelerometer in the 3D scanning algorithm.

3. Surfaces with normals parallel to the vertical y-axis are either the floor or the ceiling. In the coordinate system of the scanned mesh, the y and z axes are reversed, so the floor has a negative y component in its normal vector, while the ceiling has a positive y component.

4. Large flat surfaces perpendicular to the floor are walls. By intersecting perpendicular planes derived from the detected surfaces, we can obtain corners, wall/floor/-

ceiling intersection lines, and a bounding box for the room.

5. The resulting mesh of the 3D scan is not an exact representation of the real world due to limitations in the scanning process. To account for imprecisions, we work with ranges rather than singular values when comparing values. Additionally, the use of ranges helps mitigate the limitations of floating point number representation. For instance, when verifying if a component of a normal vector equals 1, we need to ascertain if that component falls within the range of $(x - \epsilon, x + \epsilon)$. Here, ϵ is a sufficiently small number determined by the level of noise generated by the scanning process.

   The use of ranges instead of individual values also helps circumvent the limitations associated with floating point number representation. However, in our scenario, a larger epsilon is required as the error induced by the scanning process is significantly higher than the inaccuracies inherent in floating point calculations.

For a room with a rectangular base, we can derive the orientation and size of the room from the detected corner points. In the simplest case, we need a minimum of three adjacent corners to reconstruct the base of the room. Once we have the outline of the floor and the plane of the ceiling, we can create a 3D representation of the room. For non-rectangular rooms or rooms with more complex geometries, the detection process becomes more sophisticated. Based on the gathered information, we can match CAD drawings to the floor and walls.

3 Flat Surface Detection

Using a uniform distribution random number generator, we select a specific number of faces from the mesh and apply the following methodology for each:

Firstly, all faces within a predetermined distance (measured in meters) from the original face are selected. Subsequently, we compute the average normal vector for these selected faces. The next step involves calculating the angle between the normal of the selected face and the average normal of the surrounding faces. If the resulting angle is less than a set threshold (for instance, 5 degrees), we determine that the face is a component of a flat surface and subsequently store it in an array of potential candidate faces.
This procedure is reiterated several times. With a sufficiently large sample number, we can scrutinize most of the prominent features of the mesh. Ideally, this number is determined based on the size and complexity of the mesh that is being analyzed.

4 Detection of Floor and Ceiling

During the 3D scanning process, we know the direction of the gravitational force. The resulting mesh is oriented such that the y-coordinate is parallel to and points in the same direction as the gravity vector. Using this information, we can determine if a surface is parallel to the floor or not. Surfaces with normals parallel to the y-axis and minimal nonzero x and z components can be identified as the floor and ceiling.

5 Detection of Walls

Flat surfaces with normal vectors perpendicular to the y-axis are potential walls. Similar to the method used for floor/ceiling detection, we select a certain number of eligible faces that meet our assumptions. The number of walls required may vary based on the specific room topology.

6 Detailed Description of Wall Detection Algorithm

6.1 Key Constants & Variables:

6.1.1 Wall structure

- point
- normal
- type (floor/ceiling/sidewall)

6.1.2 Constants

- starting number of random faces ($\sim$500)
- distance threshold (1m)
- threshold value for normal vector deviation between selected face and the average of the surrounding faces ($\sim$5 degrees)
- minimum number of walls to detect

6.1.3 Variables

- array of random faces
- array of wall candidates

6.2 Detection of Flat Surfaces

For performance reasons, instead of examining every face in the mesh, we randomly select a subset of faces and analyze their vicinity. The number of randomly selected faces is calculated based on the following formula:

$$N_r = bias * ln(N), \text{ where } N := \text{number of faces.}$$

For our dataset we picked *bias* to be 50, determined heuristically.

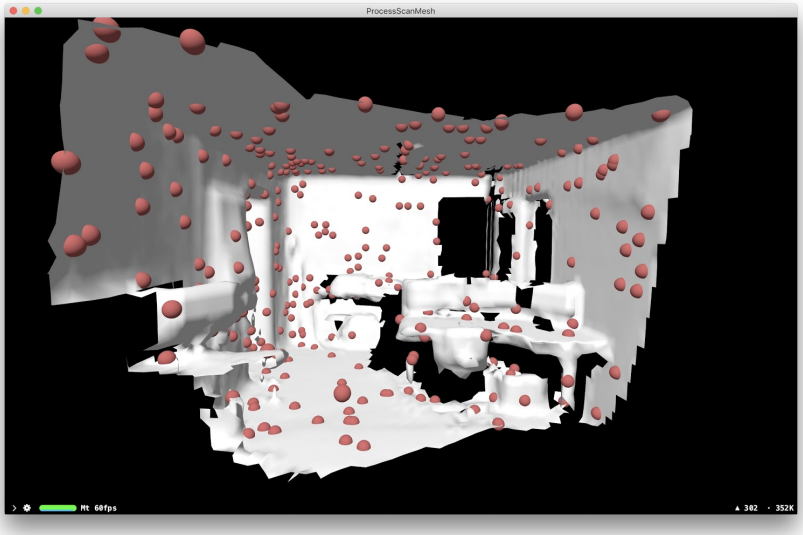

Figure 1: Randomly selected faces visualized with spheres

Once we have a starting face, we select a set of neighboring faces within a certain proximity. We calculate the distance between the selected face $\overrightarrow{f_0} = (x_0, y_0, z_0)$ and each neighboring face $\overrightarrow{f} = (x, y, z)$ using the formula for the Euclidean distance: $d = \sqrt{(x_0 - x)^2 + (y_0 - y)^2 + (z_0 - z)^2}$. If the distance falls below a specified threshold, we add the face to an array of neighboring faces.

If the specified value is below a certain threshold, the corresponding face is included in an array of adjacent faces. Since we work in 3D space, the selected faces will be within a sphere of radius r. The determination of an optimal threshold value for distance is critical for accurate wall detection. If the threshold value is too small, it may lead to the inclusion of small flat surfaces that do not represent walls (such as doors). Conversely, a threshold value that is too large may result in the omission of actual walls.

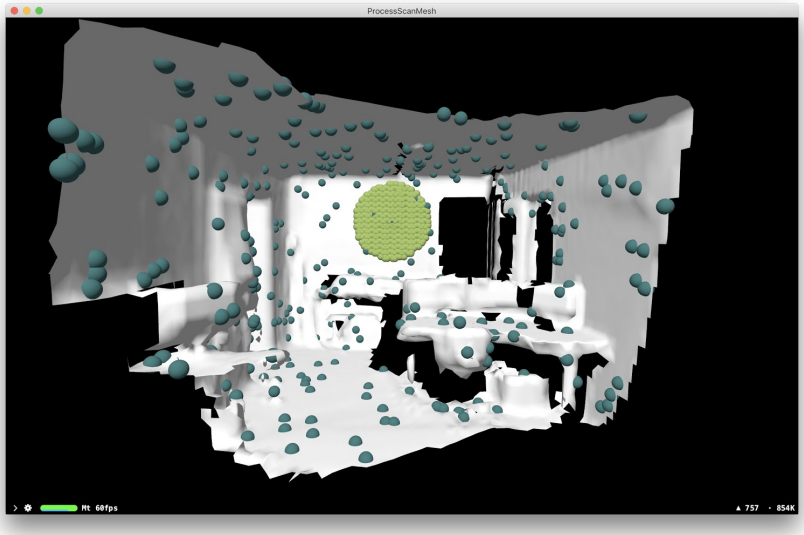

Figure 2: A face and its neighboring faces. This face will be nominated as a wall candidate.

Once we have an array of N number of normals for each neighboring face $A_{normals} = \{\overrightarrow{n}_1, ..., \overrightarrow{n}_{N-1}\}$, we calculate the average normal for them:

$$\overrightarrow{n}_{average} = \left(\sum_{i=0}^{N-1} \overrightarrow{n}_i\right) \cdot \frac{1}{N}$$

Let's have a function $f : (\mathbb{R}^3, \mathbb{R}^3) \rightarrow \mathbb{R}$ that calculates the angle between two vectors, based on the properties of the dot product:

$$f(\overrightarrow{v}_1, \overrightarrow{v}_2) = \arccos\left(\frac{\overrightarrow{v}_1 \cdot \overrightarrow{v}_2}{|\overrightarrow{v}_1||\overrightarrow{v}_2|}\right) \cdot \frac{180}{\pi}$$

Using f we calculate the angle between the average normal $\overrightarrow{n}_a$ and the center face $\overrightarrow{n}_0$:

$$\alpha = f(\overrightarrow{n}_a, \overrightarrow{n}_0) = arccos(\frac{\overrightarrow{n}_a \cdot \overrightarrow{n}_0}{|\overrightarrow{n}_a||\overrightarrow{n}_0|}) \cdot \frac{180}{\pi}$$

We also compute the maximum angle between the center face and the rest of the faces in the array of neighboring faces using function f:

$$\beta = max f(\overrightarrow{n}_0, \overrightarrow{n}_i), where \overrightarrow{n}_i \in A_{normals}.$$

Let us select a parameter T_{nd} (normal deviation threshold) for α, and then establish a condition for acceptance of a chosen center face and its peripheral area as a wall candidate. If $\alpha < T_{nd}$ and $\beta < T_{nd} * 3$, place the corresponding center face into a collection of probable faces.

The experiments were conducted on several distinct meshes, revealing that the optimal angle threshold seems to fall within the 4.0-5.0-degree range under normal conditions, specifically where the mesh does not have any significant distortions or noise. Selecting the appropriate threshold value is vital because the 3D scanning process often does not produce perfectly flat wall surfaces. The further the object is during the scanning process, the more noise gets introduced into the resulting mesh, a factor that needs to be accommodated for.

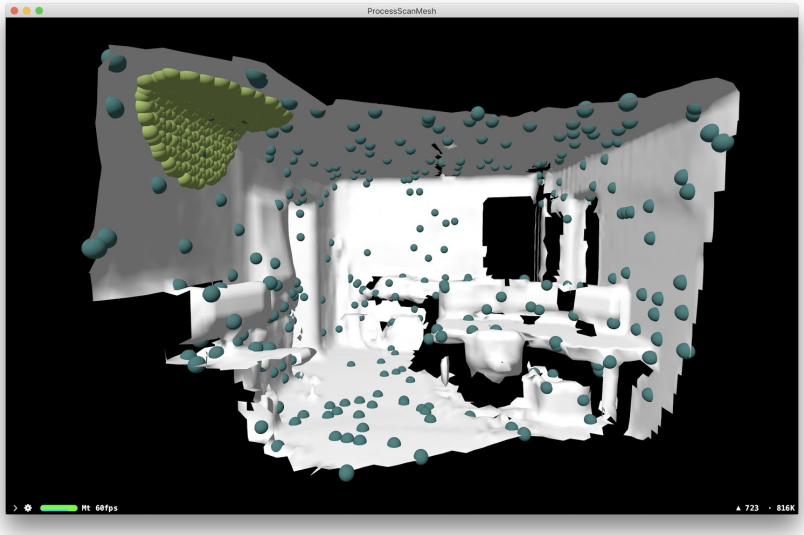

Figure 3: A face and its neighboring faces. The average normal deviation is too big for this area to be accepted.

Upon identifying a plane, we proceed to store it in an array designated for potential walls. The only necessary information to maintain is a normalized normal vector, alongside a point on the surface. This data is sufficient to mathematically formulate a plane that coincides with the surface.

7 Detecting individual flat surfaces

Our devised methodology occasionally results in multiple detections of the same plane. This occurrence leads to an array of candidate faces potentially encompassing multiple faces derived from an identical plane. Hence, a necessity arises to condense these repeated faces down to a singular instance for each distinct plane.

The implementation of this reduction involves a recursive procedure outlined below:

1. Start with the normal of the first item in the candidate faces array. Calculate the

angle against each of the other normals encapsulated within the array. Save these calculated angle/face pairs in a distinct array. If a certain plane is described by multiple faces, this will lead to certain normals exhibiting minuscule differences relative to other normals. The assumption is that these normals correspond to faces residing on the same plane.

2. If multiple faces are detected on a single plane, we select those with small normal differences and calculate two characteristics: the normal average of the faces and the center points' average.

3. Generate a new element within a planes array. This plane correlates with an individual wall, floor, or ceiling, as per the scenario.

4. Remove the selected faces from the candidate faces database and repeat the entire process using the narrowed down selection until no further candidate faces remain.

8 Plane Categorization

The following step involves categorizing the candidate facets into floor, ceiling, or sidewall groups. This requires analyzing the normals of the planes and leveraging assumption #2 and #3 to identify which planes represent the floor and ceiling. The planes not categorized will be classified as sidewalls.

After completing this procedure, a selection of potential walls remains. Based on pre-established standards, we either approve or reject the resultant group of walls. For instance, acceptable criteria might stipulate that the final set must comprise exactly one floor, one ceiling, and a minimum of one to a maximum of four vertical walls. If the requirements are not fulfilled, we return to the beginning of the search procedure and

continue until we have an accepted group of walls or until we exhaust the maximum number of allowed search iterations.

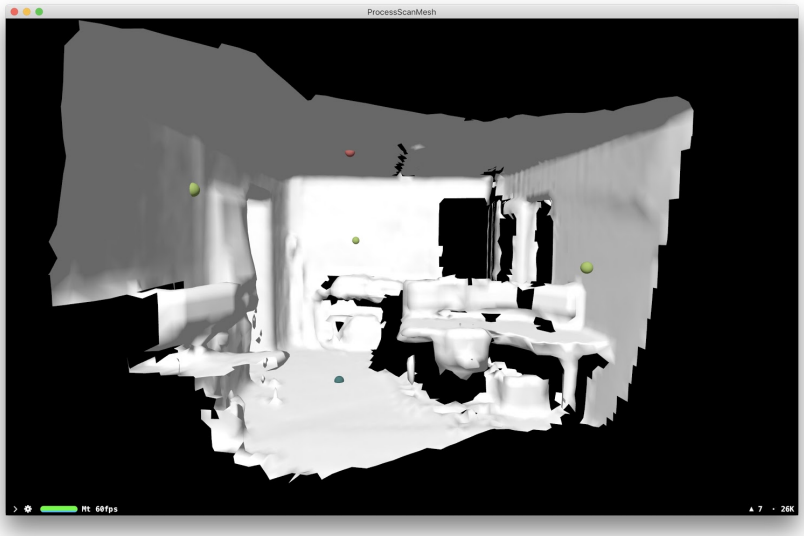

Figure 4: Mesh with colored markers on detected walls

9 Constructing 3D Plane Meshes for Identified Walls

With the available set of specified wall details (comprising normal vectors, locations, and types), the next step is to build plane meshes to visually represent these walls.

To accomplish this, a plane mesh needs four vertices to dictate both the size and orientation of the plane. The data required to build these vertices, a point on the plane and its normal vector, are both derived from the wall detection algorithm.

In order to create the plane meshes for the floor and ceiling, we leverage certain assumptions declared earlier in this document. It is assumed that the floor and ceiling are parallel to the 'xz' plane, consequently, the y coordinate remains the same for all points on these planes.

To build the vertices from a single point, it's necessary to traverse the plane in four

distinct directions. This method ensures that the initial point becomes the central point of the 3D plane mesh.

If the desired planes parallel the planes dictated by the axes of the coordinate system, the traversal is straightforward: there is only a need to add or subtract the targeted size from the coordinates of the initial point, while keeping a chosen coordinate component unchanged. For instance, on the floor plane, one traversal could be demonstrated as follows:

```
v0 = Vector3(wall.point.x + size, wall.point.y, wall.point.z + size)
v1 = Vector3(wall.point.x + size, wall.point.y, wall.point.z - size)
v2 = Vector3(wall.point.x - size, wall.point.y, wall.point.z + size)
v3 = Vector3(wall.point.x - size, wall.point.y, wall.point.z - size)
```

In contrast, the procedure for the sidewalls is considerably more complex. It cannot be assumed that the wall aligns precisely with the coordinate system's planes, hence there is no specific coordinate component retained intact throughout the process of the plane vertices' generation. A plausible solution to circumvent this issue involves rotating the plane around the y-axis until it matches one of the aforementioned coordinate system's planes, followed by employing the same traversal we use for the flooring and ceiling. Once this phase is complete, the plane's vertices are then reoriented back to their original position.

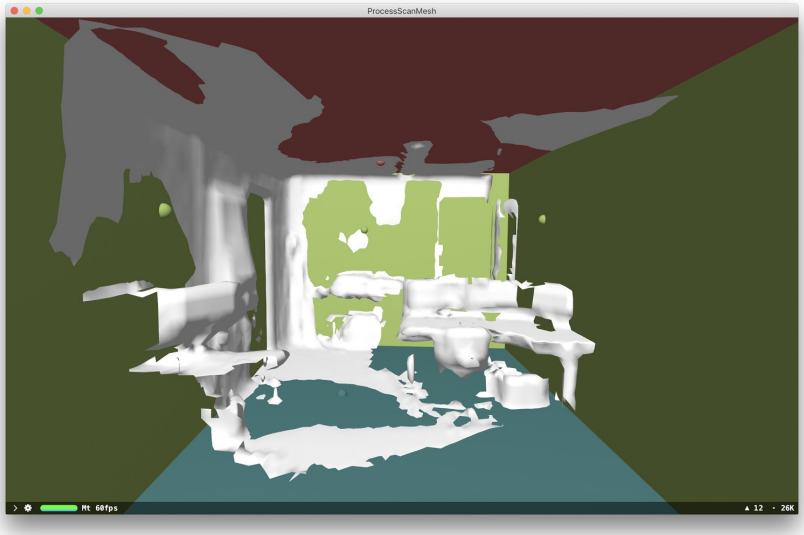

Figure 5: Mesh and detected wall planes visualized.

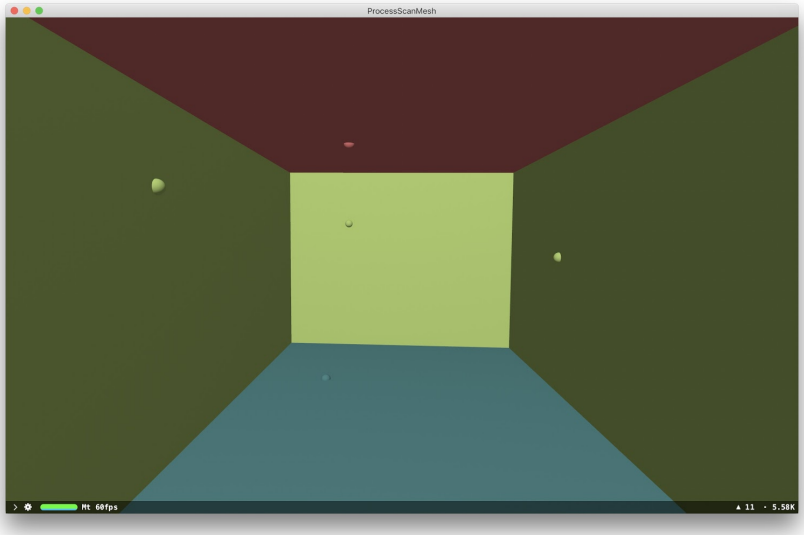

Figure 6: Detected wall planes and wall markers without mesh

10 Wall Intersection Lines

In numerous aspects, wall intersection lines are integral components within our application. These lines not only significantly contribute to an AR visualization by delineating the layout of the rooms, but they also facilitate the crucial process of discerning the underlying structure.

Figure 7: Intersection lines shown behind the mesh

Before we discuss the analytical algorithm, we describe the data structures used in the process:

10.0.1 Point

- Position coordinates
- Two lines that define the intersection point
- Three planes that define the intersection point

10.0.2 Line

- Direction vector
- An arbitrary point on the line
- Endpoint #1, Endpoint #2 (optionals)
- Two planes that define the intersection line

10.0.3 Plane

- Normal vector
- Arbitrary point on the plane
- Type: floor/ceiling/side-wall

10.1 Geometric requirements

Depending on the scan, the detected planes might intersect in different ways. We have to account for every combination.

To be able to infer the topology, we are going to set up a hard requirement in terms of plane configurations: the array of detected planes must contain a floor and at least two non-parallel (intersecting) side walls. Even if this requirement is satisfied, only a single corner point is guaranteed to be found, which is not sufficient to be able to determine the dimensions of the floor plan. For that we need at least three corner points.

10.2 Visualization of intersection lines

Once we have two intersecting planes, we can draw the intersection line in the AR view. The visualization of such intersecting lines presents a challenge, though. Formally,

an intersection line is purely mathematical - a construct that extends infinitely in two directions. This becomes problematic when we attempt to translate this into an AR environment. When presented in first-person view, such an "infinite" line doesn't seamlessly integrate with the overall scene, often resulting in a perception that feels "off" or disjointed. The solution herein lies in creating a sightline endpoint perceivable by the camera acutely, thereby facilitating a correct geometric perception.

Therefore we introduce another requirement: a visible line must have at least one endpoint. In this case the direction vector must point in the direction of the unknown other endpoint.

Even when we discuss infinite or partially infinite lines, in the realm of computer graphics there is no such thing. When we draw a line with only one known endpoint, we construct the other endpoint by traversing the line along the direction vector by an amount that exceeds the guessed room size.

10.3 Intersection strategies

When examining the intersection of the detected wall planes, it is crucial to develop a methodology for identifying plane intersections and recording the outcomes in a format that accurately represents the room's geometry.

To achieve this, we will perform line intersection calculations between all lines. Some lines will not intersect, while others will yield intersection points, which will serve as the corners.

10.4 Mapping points to planes

Depending upon the specific configuration of the detected planes, we will find a varying number of points. Our next step is to cycle through each plane, associating the points

found on the plane with that plane from the point array. Each plane's structure will then contain an array of intersection points or corners.

For a plane with two detected corner points, we require knowledge of the corresponding wall's dimensions. This information is crucial for fitting a CAD drawing onto it. Additionally, it's necessary to determine which side of the floor the detected intersection line between the corners refers to.

Upon detecting three corner points, we can deduce the width and height of the specific wall, floor, or ceiling in question. Additionally, we can identify the location of the fourth corner.

In the scenario where we find four corner points on a plane, the texture's application should be unambiguous.

10.5 Matching corner points on opposite sides

Let's have a detected topology where, say, we have the following setup:

- Floor detected
- Ceiling detected
- Two adjacent side walls detected

10.6 Possible intersection scenarios

The scenarios outlined below are based on the assumption that the following conditions hold true:

- The existence of a floor plane
- The presence of a minimum of two side-walls

In the event these conditions are not met, the user will be prompted to conduct a new scan.

10.6.1 Two Side-Walls Scenario

Parallel side-walls In this scenario, the side-walls do not intersect as they are non-adjacent walls, making the identification of corner points impossible. The intersection routine will consequently return nil if no intersection is detected.

Intersecting side-walls

Absence of a Ceiling In this particular arrangement, a corner point along with three associated lines are defined. Each of these three lines is characterized by having a distinct endpoint.

With ceiling Under this configuration, two corner points and five corresponding lines are established. The line that joins the corners is fully defined, possessing endpoints at both extremities. However, the remaining lines are only partially defined, having just one endpoint. In the opposite direction, these lines extend indefinitely.

10.6.2 Three side-walls

Given a room with a rectangular base, this setup ensures that the three side-walls intersect to form a U shape: two of them are parallel while the third intersects the other two.

Absence of a Ceiling This configuration parallels the one with two side-walls and a combination of floor and ceiling, where two corner points and five lines are similarly delineated.