MMBasic DRAW3D Graphics User Manual

Overview

The DRAW3D graphics system in PicoMite MMBasic provides commands and functions for creating, manipulating, and displaying three-dimensional objects on a 2D screen. The system uses quaternion-based rotation and supports features like:

- Multiple 3D objects (up to 8 objects)
- Multiple cameras (up to 3 cameras)
- Face-based rendering with depth sorting
- Surface normal calculations for hidden face removal
- Lighting and ambient effects
- Face flags for customization

DRAW3D CREATE

Creates a new 3D object in memory.

Syntax

```
DRAW3D CREATE n, nv, nf, camera, vertex(), facecount(), faces(),
              colours() [, linecolour()] [, fillcolour()]
```

Parameters

Parameter	Description
n	Object number (1 to 8)
nv	Number of vertices (minimum 3)
nf	Number of faces (minimum 1)
camera	Camera number to use for this object (1 to 3)
vertex()	2D array of vertex coordinates (nv x 3: x, y, z)
facecount()	1D array: number of vertices for each face
faces()	1D array listing vertex indices for each face
colours()	1D array of color values used by the object
linecolour()	Optional: 1D array of color indices for edges
fillcolour()	Optional: 1D array of color indices for fills

Notes:
- The object is stored in memory and must be closed when no longer needed.
- The camera must be configured before creating objects that reference it.
- Vertices are normalized to unit vectors internally, with magnitude stored separately.
- Centroids and surface normals are calculated automatically for each face.

DRAW3D CAMERA

Configures a camera for 3D projection.

Syntax

```
DRAW3D CAMERA n, viewplane [, x] [, y] [, panx] [, pany]
```

Parameters

Parameter	Description
n	Camera number (1 to 3)
viewplane	Distance from camera to view plane (affects perspective)
x	Optional: Camera X position (default 0, range -32766 to 32766)
y	Optional: Camera Y position (default 0, range -32766 to 32766)
panx	Optional: Horizontal pan offset (default 0)
pany	Optional: Vertical pan offset (default 0)

Example

```
DRAW3D CAMERA 1, 500
DRAW3D CAMERA 1, 400, 100, 50, -20, 10
```

DRAW3D SHOW

Displays a 3D object on screen, clearing any previous display of that object.

Syntax

```
DRAW3D SHOW n, x, y, z [, nonormals] [, depthmode]
```

Parameters

Parameter	Description
n	Object number (1 to 8)
x	X position in 3D space (range -32766 to 32766)
y	Y position in 3D space (range -32766 to 32766)
z	Z position in 3D space (distance from camera)
nonormals	Optional: 0 = hidden face removal (default), 1 = show all
depthmode	Optional: 0 = centroid depth, 1 = vertex depth, 2 = hidden-line wireframe (rp2350 only)

Example

```
DRAW3D SHOW 1, 0, 0, 200
DRAW3D SHOW 1, 0, 0, 200, 1     ' Show all faces
DRAW3D SHOW 1, 0, 0, 200, 0, 1  ' Vertex-based depth
DRAW3D SHOW 1, 0, 0, 200, 0, 2  ' Hidden-line wireframe
```

Notes:
- depthmode=2 is for closed solid meshes rendered as wireframe.
- depthmode=2 is available on rp2350 builds only.
- depthmode=2 requires a framebuffer or memory-mapped display target.
- Use no-fill faces (wireframe) for hidden-line edge rendering.
- Hidden-line mode is slower than standard solid rendering.

DRAW3D WRITE

Displays a 3D object on screen without clearing the previous display. Same parameters as DRAW3D SHOW.

Syntax

```
DRAW3D WRITE n, x, y, z [, nonormals] [, depthmode]
```

Note: depthmode=2 hidden-line wireframe is supported on rp2350 builds, with the same constraints as DRAW3D SHOW.

DRAW3D ROTATE

Rotates one or more 3D objects using a quaternion.

Syntax

```
DRAW3D ROTATE quaternion(), n1 [, n2, n3, ...]
```

Parameters

Parameter	Description
quaternion()	1D array of 5 elements: w, x, y, z, m
n1, n2, ...	Object numbers to rotate (1 to 8)

Example

```
Dim float quat(4)
' Use MATH Q_CREATE to build rotation quaternion
Math q_create Rad(10), 0, 1, 0, quat()  ' 10 deg around Y
DRAW3D ROTATE quat(), 1
```

Notes:
- Quaternion rotation avoids gimbal lock.
- Rotation is cumulative; use DRAW3D RESET to restore original orientation.
- Use MATH Q_CREATE angle, x, y, z, q() to create quaternions easily.

DRAW3D RESET

Resets one or more 3D objects to their original (pre-rotation) state.

Syntax

```
DRAW3D RESET n1 [, n2, n3, ...]
```

DRAW3D LIGHT

Sets the light source position and ambient lighting level for an object.

Syntax

```
DRAW3D LIGHT n, x, y, z, ambient
```

Parameters

Parameter	Description
n	Object number (1 to 8)
x	Light source X position (-32766 to 32766)
y	Light source Y position (-32766 to 32766)
z	Light source Z position (-32766 to 32766)
ambient	Ambient light level (0 to 100, as percentage)

Example

```
DRAW3D LIGHT 1, -100, 100, -50, 30
```

Note: The face must have flag bit 3 set (value 8) for lighting to be applied.

DRAW3D SET FLAGS

Sets rendering flags for specific faces of a 3D object.

Syntax

```
DRAW3D SET FLAGS n, flag, face1, count1 [, face2, count2, ...]
```

Parameters

Parameter	Description
n	Object number (1 to 8)
flag	Flag value (0 to 255)
face	Starting face index
count	Number of faces to set

Flag Bits

Bit	Value	Description
0	1	Hide face (don't draw)
1	2	Debug: draw face in red
2	4	Invert normal (for inside-out faces)
3	8	Apply lighting calculations

Example

```
DRAW3D SET FLAGS 1, 8, 0, 6   ' Enable lighting for all 6 faces
```

DRAW3D HIDE

Hides one or more 3D objects by clearing their display area.

Syntax

```
DRAW3D HIDE n1 [, n2, n3, ...]
```

DRAW3D HIDE ALL

Hides all currently displayed 3D objects.

Syntax

```
DRAW3D HIDE ALL
```

DRAW3D RESTORE

Restores (redraws) previously hidden 3D objects at their last displayed position.

Syntax

```
DRAW3D RESTORE n1 [, n2, n3, ...]
```

Note: An error occurs if the object is not currently hidden.

DRAW3D DIAGNOSE

Outputs diagnostic information about a 3D object's faces.

Syntax

```
DRAW3D DIAGNOSE n, x, y, z [, sort]
```

Parameters

Parameter	Description
n	Object number (1 to 8)
x	X position in 3D space
y	Y position in 3D space
z	Z position in 3D space
sort	Optional: 0 = unsorted, 1 = sorted by depth (default)

Output for each face:
- Face number and distance from camera
- Dot product (surface normal vs camera ray)
- Whether face is "Hidden" or "Showing"

DRAW3D CLOSE

Closes one or more 3D objects and frees their memory.

Syntax

```
DRAW3D CLOSE n1 [, n2, n3, ...]
```

DRAW3D CLOSE ALL

Closes all 3D objects and frees all associated memory.

Syntax

```
DRAW3D CLOSE ALL
```

DRAW3D(Functions

Functions to query 3D object properties. All return numeric values.

Bounding Box Functions

```
x = DRAW3D(XMIN n)   ' Minimum X of bounding box
x = DRAW3D(XMAX n)   ' Maximum X of bounding box
y = DRAW3D(YMIN n)   ' Minimum Y of bounding box
y = DRAW3D(YMAX n)   ' Maximum Y of bounding box
```

Position Functions

```
x = DRAW3D(X n)      ' Last displayed X position
y = DRAW3D(Y n)      ' Last displayed Y position
z = DRAW3D(Z n)      ' Last displayed Z position
```

Distance Function

```
d = DRAW3D(DISTANCE n)  ' Average distance from camera to all faces
```

Useful for collision detection or determining relative object positions.

Technical Notes

Coordinate System

- X axis: Positive to the right
- Y axis: Positive upward
- Z axis: Positive toward the camera (away from screen)

Quaternion Format

The quaternion array has 5 elements:
- q(0) = w (scalar/real component)
- q(1) = x (i component)
- q(2) = y (j component)
- q(3) = z (k component)
- q(4) = m (magnitude, typically 1.0)

Use MATH Q_CREATE angle, ax, ay, az, q() to create a quaternion for rotation of 'angle' radians around the axis (ax, ay, az).

Vertex Ordering and Surface Normals

CRITICAL: The order in which vertices are listed for each face determines the direction of the surface normal, which is essential for correct rendering.

Surface Normal Calculation:
The surface normal is calculated using the cross product of two edge vectors. For a face with vertices V0, V1, V2, the normal is computed as: (V1-V0) x (V2-V0). This follows the right-hand rule - if you curl your fingers from the first edge to the second, your thumb points in the normal direction.

Counter-Clockwise Winding:
When vertices are listed in counter-clockwise order (as viewed from outside the object), the surface normal points outward. This is the standard convention and is required for correct hidden face removal.

Why This Matters:
1. HIDDEN FACE REMOVAL: The renderer compares the surface normal to the camera direction. If the normal points away from the camera (dot product > 0), the face is back-facing and hidden. Wrong vertex order = wrong normal = faces appear/disappear incorrectly.

2. LIGHTING: Light calculations use the surface normal to determine brightness. Wrong normals cause faces to be lit from the wrong side or appear black.

3. INSIDE-OUT OBJECTS: If all faces have reversed normals, the object appears inside-out (you see the interior instead of exterior).

Fixing Reversed Faces:
- Reverse the vertex order in your face data (e.g., 0,1,2,3 becomes 3,2,1,0)
- Or use DRAW3D SET FLAGS with bit 2 (value 4) to invert the normal for specific faces

Hidden Face Removal

Faces are hidden when the dot product of the surface normal and the camera ray is positive, meaning the face is pointing away from the camera. This only works correctly when vertex ordering follows the counter-clockwise convention described above.

Depth Sorting

Two modes are available:
- Centroid mode (0): Uses distance from camera to face centroid
- Vertex mode (1): Uses maximum distance from camera to any vertex

Memory Management

- Each 3D object allocates memory for vertices, faces, colors, normals, etc.
- Always use DRAW3D CLOSE or DRAW3D CLOSE ALL when finished to free memory.
- Maximum 8 simultaneous objects and 3 cameras.

Error Messages

Error	Cause
Object already exists	Creating object with ID already in use
Minimum of 3 vertices	Creating object with < 3 vertices
Minimum of 1 face	Creating object with no faces
Vertex count < 3 for face	Face defined with < 3 vertices
Camera position not defined	Displaying before camera setup
Object % is not hidden	Restoring non-hidden object
Edge colour Index %	Color index out of range
Fill colour Index %	Color index out of range

Complete Example: Rotating Icosidodecahedron

About This Example

This example demonstrates rendering and animating a complex 3D polyhedron called an icosidodecahedron. This Archimedean solid has:

- 60 vertices
- 32 faces (12 regular pentagons + 20 regular hexagons)
- 60 edges

The program showcases several key features of the DRAW3D system:

1. VERTEX DEFINITION: Vertices are defined using the golden ratio (phi) to create mathematically precise coordinates for the polyhedron.

2. FACE DEFINITION: Faces are defined by listing vertex indices. Pentagons have 5 vertices each; hexagons have 6. The face vertex counts array tells DRAW3D how many vertices belong to each face.

3. COLOR MAPPING: A palette of 3 colors is defined (red, white, black). Each face references these by index - pentagons use red fill, hexagons use white fill, all edges use black.

4. DOUBLE BUFFERING: The FRAMEBUFFER commands enable smooth animation by drawing to an off-screen buffer, then copying to the display.

5. QUATERNION ROTATION: MATH Q_CREATE generates rotation quaternions. The parameters define the rotation angle (in radians) and the axis vector (x, y, z components).

Program Flow

1. Define vertex coordinates using DATA statements with golden ratio formulas
2. Define face connectivity (which vertices form each face)
3. Read and scale vertices to fit the screen
4. Set up color palette and face-to-color mappings
5. Create the 3D object with DRAW3D CREATE
6. Configure the camera with DRAW3D CAMERA
7. Set up double buffering with FRAMEBUFFER
8. Animation loop: rotate, reset accumulated rotation, redraw, copy buffer

Program Code

```
Option explicit            ' Require all variables to be declared
Option default none        ' No default variable type

' ==============================================================
' GOLDEN RATIO - fundamental constant for this polyhedron
' phi = (1 + sqrt(5)) / 2 = 1.618033988...
' ==============================================================
Dim float phi=(1+Sqr(5))/2

' ==============================================================
' VERTEX DATA - 60 vertices for icosidodecahedron
' Coordinates use golden ratio to create precise geometry
' Each line: x, y, z coordinates for vertices
' ==============================================================
Data 0,1,3*phi, 0,1,-3*phi, 0,-1,3*phi, 0,-1,-3*phi
Data 1,3*phi,0, 1,-3*phi,0, -1,3*phi,0, -1,-3*phi,0
Data 3*phi,0,1, 3*phi,0,-1, -3*phi,0,1, -3*phi,0,-1
Data 2,(1+2*phi),phi, 2,(1+2*phi),-phi
Data 2,-(1+2*phi),phi, 2,-(1+2*phi),-phi
Data -2,(1+2*phi),phi, -2,(1+2*phi),-phi
Data -2,-(1+2*phi),phi, -2,-(1+2*phi),-phi
Data (1+2*phi),phi,2, (1+2*phi),phi,-2
Data (1+2*phi),-phi,2, (1+2*phi),-phi,-2
Data -(1+2*phi),phi,2, -(1+2*phi),phi,-2
Data -(1+2*phi),-phi,2, -(1+2*phi),-phi,-2
Data phi,2,(1+2*phi), phi,2,-(1+2*phi)
Data phi,-2,(1+2*phi), phi,-2,-(1+2*phi)
Data -phi,2,(1+2*phi), -phi,2,-(1+2*phi)
Data -phi,-2,(1+2*phi), -phi,-2,-(1+2*phi)
Data 1,(2+phi),2*phi, 1,(2+phi),-2*phi
Data 1,-(2+phi),2*phi, 1,-(2+phi),-2*phi
Data -1,(2+phi),2*phi, -1,(2+phi),-2*phi
Data -1,-(2+phi),2*phi, -1,-(2+phi),-2*phi
Data (2+phi),2*phi,1, (2+phi),2*phi,-1
Data (2+phi),-2*phi,1, (2+phi),-2*phi,-1
Data -(2+phi),2*phi,1, -(2+phi),2*phi,-1
Data -(2+phi),-2*phi,1, -(2+phi),-2*phi,-1
Data 2*phi,1,(2+phi), 2*phi,1,-(2+phi)
Data 2*phi,-1,(2+phi), 2*phi,-1,-(2+phi)
Data -2*phi,1,(2+phi), -2*phi,1,-(2+phi)
Data -2*phi,-1,(2+phi), -2*phi,-1,-(2+phi)
```

```
' ==============================================================
' FACE DATA - 32 faces total
' First 12 faces: pentagons (5 vertices each)
'
' IMPORTANT: Vertex indices must be listed in COUNTER-CLOCKWISE
' order when viewing the face from OUTSIDE the object.
' This ensures surface normals point outward for correct:
'   - Hidden face removal (back faces not drawn)
'   - Lighting calculations (faces lit from correct side)
' Wrong order = faces appear/disappear incorrectly!
' ==============================================================
Data 0,28,36,40,32, 33,41,37,29,1, 34,42,38,30,2
Data 3,31,39,43,35, 4,12,44,45,13, 15,47,46,14,5
Data 17,49,48,16,6, 7,18,50,51,19, 8,20,52,54,22
Data 23,55,53,21,9, 26,58,56,24,10, 25,57,59,27,11

' Next 20 faces: hexagons (6 vertices each)
' Same rule: counter-clockwise from outside view
```

```
Data 32,56,58,34,2,0, 0,2,30,54,52,28, 29,53,55,31,3,1
Data 1,3,35,59,57,33, 13,37,41,17,6,4, 4,6,16,40,36,12
Data 5,7,19,43,39,15, 14,38,42,18,7,5, 22,46,47,23,9,8
Data 8,9,21,45,44,20, 10,11,27,51,50,26, 24,48,49,25,11,10
Data 36,28,52,20,44,12, 13,45,21,53,29,37, 14,46,22,54,30,38
Data 39,31,55,23,47,15, 16,48,24,56,32,40, 41,33,57,25,49,17
Data 42,34,58,26,50,18, 19,51,27,59,35,43
```

```
' ==============================================================
' VARIABLE DECLARATIONS
' ==============================================================
Dim float q1(4)             ' Quaternion for rotation (w,x,y,z,m)
Dim integer i, j            ' Loop counters
Dim integer nf=32           ' Number of faces
Dim integer nv=60           ' Number of vertices
Dim integer camera=1        ' Camera number to use
Dim float vertices(2,59)    ' Vertex array: 3 coords x 60 vertices

' ==============================================================
' READ AND SCALE VERTICES
' Scale so object fits nicely on screen (50% of vertical res)
' ==============================================================
For j=0 To 59               ' Loop through all 60 vertices
  For i=0 To 2              ' Read x, y, z for each
    Read vertices(i,j)
  Next i
Next j
' Scale all vertices: divide by max value, multiply by half screen
Math scale vertices(), MM.VRES/Math(max vertices())*0.5, vertices()
```

```
' ==============================================================
' READ FACE CONNECTIVITY DATA
' 12 pentagons x 5 + 20 hexagons x 6 = 180 vertex indices
' ==============================================================
Dim integer faces(179)      ' Array for all face vertex indices
For i=0 To 179
  Read faces(i)
Next i

' ==============================================================
' FACE VERTEX COUNTS
' First 12 faces have 5 vertices (pentagons)
' Next 20 faces have 6 vertices (hexagons)
' ==============================================================
Dim integer fc(nf-1) = (5,5,5,5,5,5,5,5,5,5,5,5,
                        6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6)

' ==============================================================
' COLOR PALETTE
' Index 0 = Red (for pentagon fills)
' Index 1 = White (for hexagon fills)
' Index 2 = Black (for all edges)
' ==============================================================
Dim integer colours(2) = (RGB(red), RGB(white), RGB(black))

' Edge colors: all faces use color index 2 (black)
Dim integer edge(nf-1) = (2,2,2,2,2,2,2,2,2,2,2,2,
                          2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2)

' Fill colors: pentagons=0 (red), hexagons=1 (white)
Dim integer fill(nf-1) = (0,0,0,0,0,0,0,0,0,0,0,0,
                          1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
```

```
' ==============================================================
' CREATE 3D OBJECT AND SET UP CAMERA
' ==============================================================
' Create initial rotation quaternion (small rotation to start)
Math q_create Rad(2), 1, 0.5, 0.25, q1()

' Create the 3D object with all geometry and color data
' Parameters: object#, vertices, faces, camera, vertex data,
'              face counts, face indices, colors, edge colors, fills
Draw3D create 1, nv, nf, camera, vertices(), fc(), faces(),
              colours(), edge(), fill()

' Set up camera: viewplane distance=800, position at origin
Draw3D camera 1, 800, 0, 0

' ==============================================================
' SET UP DOUBLE BUFFERING FOR SMOOTH ANIMATION
' ==============================================================
FRAMEBUFFER create         ' Create off-screen frame buffer
FRAMEBUFFER write f        ' Direct drawing to frame buffer

' Initial display of the object
Draw3D show 1, 0, 0, 1000  ' Show at position (0,0,1000)

' ==============================================================
' MAIN ANIMATION LOOP
' ==============================================================
Do
  ' Create rotation quaternion: 0.25 degrees around axis (1,3,5)
  ' This creates a tumbling motion around a diagonal axis
  Math q_create Rad(0.25), 1, 3, 5, q1()

  ' Apply rotation to object 1
  Draw3D rotate q1(), 1

  ' Reset prepares for next rotation (updates base orientation)
  Draw3D reset 1

  ' Redraw the object at same position
  Draw3D show 1, 0, 0, 1000

  ' Copy frame buffer to display (f=frame, n=now/display)
  FRAMEBUFFER copy f, n
Loop                       ' Infinite loop - press Ctrl+C to stop
```