FzxNGN Documentation

June 17,2023

Overview: fzxNGN is a 2D physics engine library that was ported to QB64 from the Impulse engine written by Randy Gaul https://github.com/RandyGaul/ImpulseEngine.

Features:

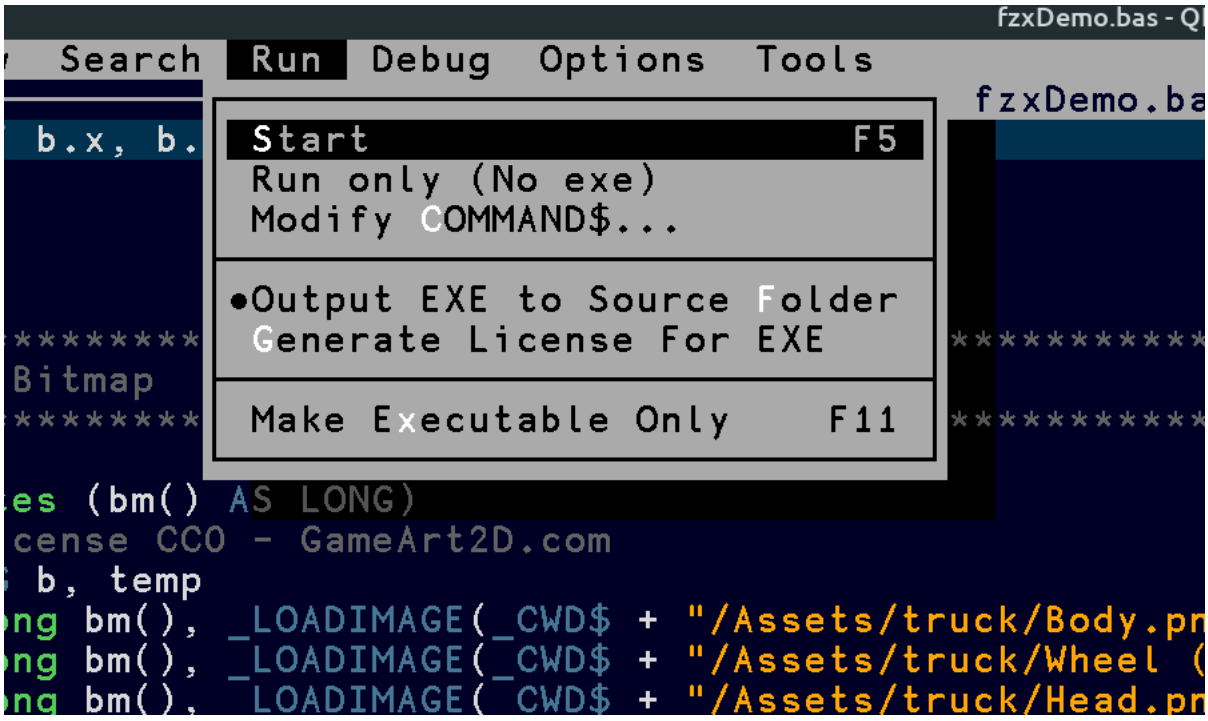
- Rigid body simulation
- Circle and polygon primitives, concave objects not supported yet
- Joint simulation
- Camera Library
- Input Library
- Finite State Machine helper functions
- Perlin noise library functions
- XML parsing (WIP)
- LERP functions
- FPS helper functions
- Tons of vector and matrix math functions.
- Units are arbitrary, Its up to the user to decide which units of length to use.

What it is and what it is not:

- A project to help QB64 programmers such as myself make more interesting demos and mini-games.
- It is a collection of subroutines and functions that I've made every attempt to generalize. They can be used outside the simulation.
- Not a core engine for the next AAA game.
- Not for serious engineering use.

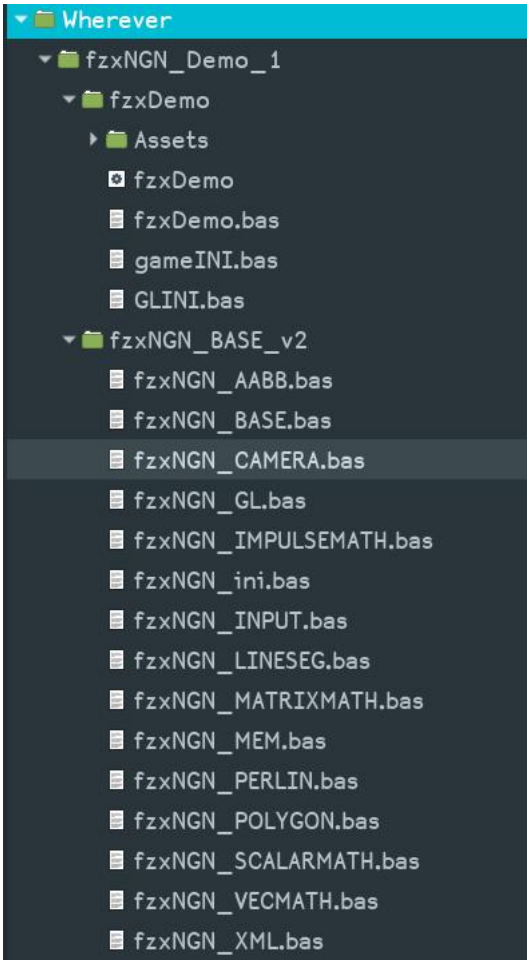
Compiling examples:

- Compiling should be straightforward. Load the example file and hit F5 or Start.
- Make sure the "Output EXE to Source Folder" is selected. This will ensure that the file structure is not broken.



FzxNGN File structure:

- The filenames may vary but the same general file structure should be maintained.



FzxNGN Globals (fzxNGN_ini.bas):

__fzxBody(): Contains all the data pertaining to each rigid body.
__fzxJoints(): Contains all the data pertaining to the joints.
__fzxHits(): Collision information
__fzxCamera : Camera data
__fzxWorld : World data
__fzxFPSCount : FPS counting
__fzxInputDevice : Mouse and keyboard
__fzxSettings : Generalized settings. (Currently only mouse double click timing)

A Bare Bone Implementation:

This is a simple example of what it takes for a small implementation of the engine.

- **Initialization include**
  - '$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'
  - sets up the types(UDT), global variables, and constants
- Call to **Build scene.**
- **Main Loop**
  - Call to **Animate scene**
    - Player interaction happens here
  - Call to fzxNGN to calculate the next step
    - **fzxImpulseStep delta time, iterations**
      - Delta Time is your time step i.e. 1/30 of a second
      - Iterations how many steps the simulation runs per step.
  - Call to **Render scene**

- **Include core code**
  - '$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'
  - All the core functionality is contained here
- **Build Scene**
  - Initial setup of the camera.
  - Set some limits on the world.
  - Setup gravity.
  - Add your bodies to the simulation.
- **Animate Scene**
  - This where you will interact with the bodies.
- **Render Scene**
  - Draw the bodies in the scene.

Code Example: fzxNGNBareBones.bas

```
'$DYNAMIC

'$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'

SCREEN _NEWIMAGE(1024, 768, 32)

DIM AS LONG iterations: iterations = 2

DIM SHARED AS DOUBLE dt: dt = 1 / 60
```

- **'$DYNAMIC**
  - Some of the Global variable are dynamic and this must be declared
- **'$include:'..\fzxNGN_BASE_v2\fzxNGN_ini.bas'**
  - Include the types, constants, and the global variables
- **SCREEN _NEWIMAGE(1024, 768, 32)**
  - Screen size can be whatever you desire, because the camera functionality will allow the user to view any portion of the world.
- **DIM AS LONG iterations: iterations = 2**
  - Number of simulation steps ran before rendering.
- **DIM SHARED AS DOUBLE dt: dt = 1 / 60**
  - Time step of simulation.

```
buildScene

DO
  CLS: LOCATE 1: PRINT "Click the mouse on the play field to spawn an object"

  fzxHandleInputDevice

  animatescene

  fzxImpulseStep dt, iterations

  renderBodies

_DISPLAY
LOOP UNTIL INKEY$ = CHR$(27)

SYSTEM

'$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'
```

- **buildScene**
  - Call to the subroutine that will setup the world and build your initial scene
- **DO**
  - This is the Main loop of the program
- **CLS: LOCATE 1: PRINT "Click the mouse on the play field to spawn an object"**
- **fzxHandleInputDevice**
  - A subroutine that will manage some extra routines for the mouse and keyboard
  - Later in the program, **"__fzxInputDevice.mouse.b1.NegEdge"** flag will be queried, It will return a true if the mouse button has been released.
- **Animatescene**
  - This is a call to the subroutine that handles all of the frame to frame activities.
  - User interaction
  - Level animation and logic
- **fzxImpulseStep dt, iterations**
  - Run the simulation
- **renderBodies**
  - Draw the level
  - This is up to the user to implement depending on their need.
- **_DISPLAY**
- **LOOP UNTIL INKEY$ = CHR$(27)**
- **SYSTEM**
- **'$include:'..\fzxNGN_BASE_v2\fzxNGN_BASE.bas'**
  - Provides all of the fzxNGN functionality

```
SUB animateScene
  DIM AS LONG temp

  IF __fzxInputDevice.mouse.b1.NegEdge THEN

    IF RND > .5 THEN

      temp = fzxCreateCircleBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10)

    ELSE

      temp = fzxCreateBoxBodyEx("b" + _TRIM$(STR$(RND * 1000000000)), 10, 10)

    END IF

    fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxInputDevice.mouse.worldPosition.x,
__fzxInputDevice.mouse.worldPosition.y

    fzxSetBody cFZX_PARAMETER_VELOCITY, temp, __fzxInputDevice.mouse.velocity.x,
__fzxInputDevice.mouse.velocity.y

    fzxSetBody cFZX_PARAMETER_ORIENT, temp, _D2R(RND * 360), 0
    fzxSetBody cFZX_PARAMETER_RESTITUTION, temp, .5, 0 ' Bounce
    fzxSetBody cFZX_PARAMETER_STATICFRICTION, temp, .1, 0
    fzxSetBody cFZX_PARAMETER_DYNAMICFRICTION, temp, .85, 0
    fzxSetBody cFZX_PARAMETER_LIFETIME, temp, RND * 20 + 10, 0
  END IF
END SUB
```

```
SUB buildScene
  DIM AS LONG temp

  __fzxCamera.zoom = 1
  fzxCalculateFOV
  fzxVector2DSet __fzxCamera.position, __fzxWorld.spawn.x, __fzxWorld.spawn.y - 300

  fzxVector2DSet __fzxWorld.minusLimit, -200000, -200000
  fzxVector2DSet __fzxWorld.plusLimit, 200000, 200000
  fzxVector2DSet __fzxWorld.spawn, 0, 0
  fzxVector2DSet __fzxWorld.gravity, 0.0, 10.0

  DIM o AS tFZX_VECTOR2d
  fzxVector2DMultiplyScalarND o, __fzxWorld.gravity, dt
  __fzxWorld.resting = fzxVector2DLengthSq(o) + cFZX_EPSILON

  temp = fzxCreateBoxBodyEx("floor", 800, 10)
  fzxSetBody cFZX_PARAMETER_POSITION, temp, __fzxWorld.spawn.x, __fzxWorld.spawn.y
  fzxSetBody cFZX_PARAMETER_STATIC, temp, 0, 0

END SUB
```

```
SUB renderBodies STATIC
  DIM i AS LONG
  DIM AS tFZX_VECTOR2d scSize, scMid, scUpperLeft, camUpperLeft, aabbUpperLeft, aabbSize, aabbHalfSize
  DIM AS LONG ub: ub = UBOUND(__fzxBody)

  fzxVector2DSet aabbSize, 40000, 40000
  fzxVector2DSet aabbHalfSize, aabbSize.x / 2, aabbSize.y / 2

  fzxVector2DSet scUpperLeft, 0, 0
  fzxVector2DSet scSize, _WIDTH, _HEIGHT

  fzxVector2DDivideScalarND scMid, scSize, 2
  fzxVector2DSubVectorND camUpperLeft, __fzxCamera.position, scMid

  i = 0: DO WHILE i < ub
    IF __fzxBody(i).enable THEN
      'fzxAABB to cut down on rendering objects out of camera view
      fzxVector2DSubVectorND aabbUpperLeft, __fzxBody(i).fzx.position, aabbHalfSize
      IF fzxAABBOverlap(camUpperLeft.x, camUpperLeft.y, scSize.x, scSize.y, aabbUpperLeft.x,
aabbUpperLeft.y, aabbSize.x, aabbSize.y) THEN
        IF __fzxBody(i).shape.ty = cFZX_SHAPE_CIRCLE THEN
          renderWireFrameCircle i, _RGB32(0, 255, 0)
        ELSE IF __fzxBody(i).shape.ty = cFZX_SHAPE_POLYGON THEN
            renderWireFramePoly i
          END IF
        END IF
      END IF
    END IF
    i = i + 1
  LOOP
END SUB
```

```
SUB renderWireFrameCircle (index AS LONG, c AS LONG)
  DIM AS tFZX_VECTOR2d o1, o2
  fzxWorldToCameraEx __fzxBody(index).fzx.position, o1
  CIRCLE (o1.x, o1.y), __fzxBody(index).shape.radius * __fzxCamera.zoom, c
  o2.x = o1.x + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * COS(__fzxBody(index).fzx.orient)
  o2.y = o1.y + (__fzxBody(index).shape.radius * __fzxCamera.zoom) * SIN(__fzxBody(index).fzx.orient)
  LINE (o1.x, o1.y)-(o2.x, o2.y), c
END SUB
```

```
SUB renderWireFramePoly (index AS LONG)
  DIM vert(3) AS tFZX_VECTOR2d

  fzxGetBodyVert index, 0, vert(0)
  fzxWorldToCamera index, vert(0)

  fzxGetBodyVert index, 1, vert(1)
  fzxWorldToCamera index, vert(1)

  fzxGetBodyVert index, 2, vert(2)
  fzxWorldToCamera index, vert(2)

  fzxGetBodyVert index, 3, vert(3)
  fzxWorldToCamera index, vert(3)

  LINE (vert(0).x, vert(0).y)-(vert(1).x, vert(1).y), _RGB(0, 255, 0)
  LINE (vert(1).x, vert(1).y)-(vert(2).x, vert(2).y), _RGB(0, 255, 0)
  LINE (vert(2).x, vert(2).y)-(vert(3).x, vert(3).y), _RGB(0, 255, 0)
  LINE (vert(3).x, vert(3).y)-(vert(0).x, vert(0).y), _RGB(0, 255, 0)
END SUB
```

Body Creation:

The following function create bodies for the simulation.

fzxCreateCircleBodyEx ("unique name for object", radius)

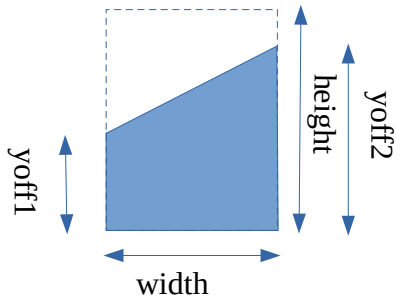
- Adds a circle body to simulation
- returns an index to the body in the __fzxBody() array

fzxCreateBoxBodyEx ("unique name for object", Width, Height)

- Adds a Rectangle to the simulation
- returns an index to the body in the __fzxBody() array

fzxCreateTrapBodyEx ("unique name for object", Width, Height, yoff1, yoff2)

- Adds a trapezoid to the simulation
- returns an index to the body in the __fzxBody() array



An example of a body creation would be like:

```
1.  box = fzxCreateBoxBodyEx("box", 100, 100)
2.  fzxSetBody cFZX_PARAMETER_POSITION, box, 100, 100
3.  fzxSetBody cFZX_PARAMETER_STATIC, box, 0, 0
4.  fzxSetBodyEx cFZX_PARAMETER_ORIENT, "box", _D2R(90), 0
```

Line 1 creates a box named "box" that is 100 units long by 100 units wide. As stated earlier units are arbitrary, so it can be 100 miles or 100 millimeters. Its up to the user to decide.

Line 2 the body is moved to a position of 100, 100. Again units are arbitrary.

Line 3 the body is set to static, and acts as a wall or a solid obstacle. You can still move it or arrange it as you see fit.

Line 4 the body is now addressed by its name instead of index and the orientation is set to 90 degrees.

Body Parameters:

The following parameters can be set by the fzxSetBody subroutine.

- fzxSetBody (Parameter, Index, argument 1, argument 2)
  - Index in the body in the __fzxBody() array you are changing
  - The arguments are the new values.
    - Argument 2 may not always be necessary. Just leave it 0.
  - Parameter Contants
    - cFZX_PARAMETER_POSITION
      - Argument 1 – X position in the world
      - Argument 2 – Y position in the world
    - cFZX_PARAMETER_VELOCITY
      - Argument 1 – X velocity in the world
      - Argument 2 – Y velocity in the world
    - cFZX_PARAMETER_FORCE
      - Argument 1 – X force applied to body
      - Argument 2 – Y force applied to body
    - cFZX_PARAMETER_ANGULARVELOCITY
      - Argument 1 – angular velocity in the world
      - Argument 2 – not used
    - cFZX_PARAMETER_TORQUE
      - Argument 1 – torque force applied to body
      - Argument 2 – not used
    - cFZX_PARAMETER_ORIENT
      - Argument 1 – body angle in radians
      - Argument 2 – not used
    - cFZX_PARAMETER_STATICFRICTION
      - Argument 1 – static friction on the body surface
      - Argument 2 – not used
      - More info https://en.wikipedia.org/wiki/Friction
    - cFZX_PARAMETER_DYNAMICFRICTION
      - Argument 1 – dynamic/kinetic friction on the body surface
      - Argument 2 – not used
      - More info https://en.wikipedia.org/wiki/Friction
    - cFZX_PARAMETER_RESTITUTION
      - Argument 1 – bounciness of the body surface
      - Argument 2 – not used
    - cFZX_PARAMETER_COLOR
      - Argument 1 – color used in wire frame, depends on renderer to implement.
      - Argument 2 – not used

- cFZX_PARAMETER_ENABLE
  - Argument 1 – 0 or non zero
    - Removes body from simulation
    - can be reenabled
  - Argument 2 – not used
- cFZX_PARAMETER_STATIC
  - Sets the object as static and object act like a wall or permanant fixture
  - Argument 1 – not used
  - Argument 2 – not used
- cFZX_PARAMETER_TEXTURE
  - Sets the texture for the body, depends on renderer to implement.
  - Argument 1 – valid texture handle.
  - Argument 2 – not used
- cFZX_PARAMETER_FLIPTEXTURE
  - Flip texture flag, depends on renderer to implement.
  - Argument 1 – 0 or non zero
  - Argument 2 – not used
- cFZX_PARAMETER_SCALETEXTURE
  - Scale texture multiplier, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_OFFSETTEXTURE
  - Shift texture by offset, depends on renderer to implement.
  - Argument 1 – X axis
  - Argument 2 – Y axis
- cFZX_PARAMETER_COLLISIONMASK
  - Used to selectively allow collisions between bodies
    - A value of &B00000001 on one body and value &B00000001 on another body will collide.
    - A value of &B00000010 on one body and &B00000001 on another body will not collide.
    - The default is &B11111111.
    - They essentially logically ANDed together.
  - Argument 1 – unsigned integer
  - Argument 2 – not used
- cFZX_PARAMETER_INVERTNORMALS
  - Experimental feature (I don’t recommend using it)
  - Argument 1 – unsigned integer
  - Argument 2 – not used
- cFZX_PARAMETER_NOPHYSICS
  - Used for sensors. Similar to cFZX_PARAMETER_ENABLE, but body still picks up collisions, but wont react to them.
  - Argument 1 – 0 or non zero
  - Argument 2 – not used

- cFZX_PARAMETER_SPECIALFUNCTION
  - User functionality, can be used for whatever the user needs
  - Argument 1 – any value
  - Argument 2 – any value
- cFZX_PARAMETER_RENDERORDER
  - Depreciated – left for compatibility
  - Argument 1 – any value
  - Argument 2 – unused
- cFZX_PARAMETER_ENTITYID
  - User functionality, can be used for whatever the user needs
  - Argument 1 – any value
  - Argument 2 – unused
- cFZX_PARAMETER_LIFETIME
  - Give the body a finite lifetime
  - Argument 1 – time in seconds
  - Argument 2 – unused
- cFZX_PARAMETER_REPEATTEXTURE
  - Repeat texture multiplier, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_ZPOSITION
  - Sets the body render order, depends on renderer to implement.
  - Argument 1 – Z axis
  - Argument 2 – unused
- cFZX_PARAMETER_UV0
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV1
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV2
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number
- cFZX_PARAMETER_UV3
  - Texture Coordinates, depends on renderer to implement.
  - Argument 1 – X axis, positive non zero number
  - Argument 2 – Y axis, positive non zero number

Querying Body Parameters:

Making this easier is on the To-Do list. All of the parameters that have been set can be read by looking at the __fzxBody() structure. The structure is defined in the fzxNGN_ini.bas file.

From the earlier example we can look at the current position

```
1. PRINT __fzxBody(box).fzx.position.x
2. PRINT __fzxBody(box).fzx.position.y
```