Octaga Nodes

1 Supported Nodes

This document specifies the nodes supported by Octaga Player 4.0 and Octaga Panorama 4.0

1.1 VRML nodes

All VRML Nodes are supported.
Specification :
http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

1.2 X3D nodes

All X3D Nodes in Edition 2 are supported except the following:

- DIS Component
- Picking Sensor Component
- NurbsSweptSurface
- NurbsSwungSurface
- GeoTransform

See the documentation of each Octaga product for a detailed list of nodes supported by specific releases.

Specification X3D (Edition 2):
http://www.web3d.org/files/specifications/19775-1/V3.2/index.html

All X3D specifications:
http://www.web3d.org/x3d/specifications/

1.1 Extended X3D nodes

Some of the nodes specified by X3D have been given extended functionality in Octaga Player by allowing additional fields or additional allowed values to already defined fields. For the following nodes the Octaga specific additional functionality is described. For the basic X3D behavior and functionality see the X3D specification.

1.1.1 BoundedPhysicsModel

BoundedPhysicsModel {

exposedField	SFBool	**enabled**	TRUE		
exposedField	SFNode	**geometry**	NULL		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFFloat	**absorption**	0	[0-1]	(Octaga Specific)
exposedField	SFFloat	**reflection**	1	[0-1]	(Octaga Specific)

}

The BoundedPhysicsModel node is identical to the standard X3D BoundedPhysicsModel node except for some additional fields. The absorption field defines the probability of the particle dying when it hits the boundary. The particle's new velocity after hitting the boundary is defined as follows: velocity = velocity - normal*(velocity* normal)*(1+ **reflection**). Thus a reflection value of 1

will result in a perfectly elastic colllision where the particle remain all its inertia while a reflection value of 0 will result in the particle sticking to the boundary like a wet cloth.

1.1.2 Collision

Collision {

eventIn	MFNode	**addChildren**		
eventIn	MFNode	**removeChildren**		
exposedField	SFBool	**enabled**		
exposedField	MFNode	**children**	[]	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFBool	**isPickable**	TRUE	(Octaga Specific)
eventOut	SFTime	**collideTime**		
eventOut	SFBool	**isActive**		
field	SFVec3f	**bboxCenter**	0 0 0	
field	SFVec3f	**bboxSize**	-1 -1 -1	
field	SFNode	**proxy**	NULL	

}

The Collision node is identical to the standard X3D Collision node except for one additional field. The **isPickable** field defines if the children nodes takes part in picking operations.

1.1.3 ComposedCubeMapTexture

ComposedCubeMapTexture {

exposedField	SFNode	**back**	NULL	
exposedField	SFNode	**bottom**	NULL	
exposedField	SFNode	**front**	NULL	
exposedField	SFNode	**left**	NULL	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFNode	**right**	NULL	
exposedField	SFNode	**top**	NULL	
field	SFNode	**textureProperties**	NULL	(Octaga Specific)

}

The ComposedCubeMapTexture node is identical to the standard X3D ComposedCubeMapTexture except for one additional field. The **textureProperties** field allows the user to set textureProperties of the complete CubeMap in the same way its done for GeneratedCubeMapTexture and ImageCubeMapTexture.

1.1.4 ComposedTexture3D

ComposedTexture3D {

exposedField	SFNode	**metadata**	NULL	
exposedField	MFNode	**texture**	[]	
field	SFBool	**repeatS**	FALSE	
field	SFBool	**repeatT**	FALSE	
field	SFBool	**repeatR**	FALSE	
field	SFNode	**textureProperties**	NULL	(Octaga Specific)

}

The ComposedTexture3D node is identical to the standard X3D ComposedTexture3D except for one additional field. The **textureProperties** field allows the user to set textureProperties of the complete Texture in the same way its done for ImageTexture3D and PixelTexture3D.

1.1.5 ConeEmitter

ConeEmitter {

exposedField	SFFloat	**angle**	PI/4		
exposedField	SFVec3f	**direction**	0 1 0		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFVec3f	**position**	0 0 0		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)

}

The ConeEmitter node is identical to the standard X3D ConeEmitter except for three additional fields. The speedVariation, massVariation and surfaceAreaVarion allows the variation for speed, mass and surfaceArea to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.1.6 ExplosionEmitter

ExplosionEmitter {

exposedField	SFNode	**metadata**	NULL		
exposedField	SFVec3f	**position**	0 0 0		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)

}

The ExplosionEmitter node is identical to the standard X3D ExplosionEmitter except for three additional fields. The speedVariation, massVariation and surfaceAreaVarion allows the variation for speed, mass and surfaceArea to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.1.7 Fog

Fog {

eventIn	SFBool	**set_bind**	
exposedField	SFColor	**color**	1 1 1
exposedField	SFString	**fogType**	“LINEAR”
exposedField	SFFloat	**visibilityRange**	0
exposedField	SFNode	**metadata**	NULL
eventOut	SFBool	**isBound**	
eventOut	SFTime	**bindTime**	

}

The Fog node is identical to the standard X3D Fog node for some extra legal values for **fogType**.

"**LINEAR**" (f = (visibilityRange-d_V) / visibilityRange)

"**EXPONENTIAL**" (f = exp(-d_V / (fogVisibility-d_V)))

"**OGLEXP**" (f = exp(-(d_V/visibilityRange)))

"**OGLEXP2**" (f = exp(-(d_V/visibilityRange)^2)))

The last two are Octaga specific.

1.1.8 FontStyle

FontStyle {

exposedField	SFNode	**metadata**	NULL	
field	MFString	**family**	[“SERIF”]	
field	SFBool	**horizontal**	TRUE	
field	MFString	**justify**	[“BEGIN”]	
field	SFString	**language**	“”	
field	SFBool	**leftToRight**	TRUE	
field	SFFloat	**size**	1.0	
field	SFFloat	**spacing**	1.0	
field	SFString	**style**	“PLAIN”	
field	SFBool	**topToBottom**	TRUE	
field	SFString	**type**	“”	(Octaga Specific)
field	SFInt32	**resolution**	-1	(Octaga Specific)

}

The FontStyle node is identical to the standard X3D FontStyle node except for some additional fields. The **type** field specifies the type of text to display. Possible values include “POLYGON” for tesselated polygonial fonts, “TEXTURE” for texture fonts and “EXTRUDED” for extruded polygon fonts. If the **type** is empty polygon fonts are used in 3d scenes and texture fonts are used in 2D scenes. The height of the extruded fonts are 1.0 (which can be scaled using a Transform node).

1.1.9 IndexedFaceSet

IndexedFaceSet {

eventIn	MFInt32	**set_colorIndex**		
eventIn	MFInt32	**set_coordIndex**		
eventIn	MFInt32	**set_normalIndex**		
eventIn	MFInt32	**set_texCoordIndex**		
exposedField	MFNode	**attrib**	[]	
exposedField	SFNode	**color**	NULL	
exposedField	SFNode	**coord**	NULL	
exposedField	SFNode	**fogCoord**	[]	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFNode	**normal**	NULL	
exposedField	SFNode	**texCoord**	NULL	
field	SFBool	**ccw**	TRUE	
field	MFInt32	**colorIndex**	[]	
field	SFBool	**colorPerVertex**	TRUE	
field	SFBool	**convex**	TRUE	
field	MFInt32	**coordIndex**	[]	
field	SFFloat	**creaseAngle**	0	
field	MFInt32	**normalIndex**	[]	
field	SFBool	**normalPerVertex**	TRUE	
field	SFBool	**solid**	TRUE	
field	MFInt32	**texCoordIndex**	[]	
field	SFBool	**sortPolygons**	TRUE	(Octaga Specific)

}

The IndexedFaceSet node is identical to the standard X3D IndexedFaceSet node except for one additional field. The sortPolygons field specifies if individual polygons should be sorted in order to provide better transparency blending.

1.1.10 KeySensor

MultiTexture {

exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
eventOut	SFInt32	**actionKeyPress**	
eventOut	SFInt32	**actionKeyRelease**	
eventOut	SFBool	**altKey**	
eventOut	SFBool	**controlKey**	
eventOut	SFBool	**isActive**	
eventOut	SFString	**keyPress**	
eventOut	SFString	**keyRelease**	
eventOut	SFBool	**shiftKey**	

}

The KeySensor node is identical to the standard X3D KeySenor node except for several added possible values returned by actionKeyPressed, and actionKeyReleased. The mappings used by Octaga is shown in the following table: (Values 1-20 are defined by X3D)

KEY	VALUE	KEY	VALUE	KEY	VALUE	KEY	VALUE
F1-F12	1-12	HOME	13	END	14	PGUP	15
PGDN	16	UP	17	DOWN	18	LEFT	19
RIGHT	20	BACK	21	TAB	22	RETURN	23
PAUSE	24	CAPS	25	ESCAPE	26	INSERT	27
DELETE	28	0-9	30-39	NUM 0-9	40-49	MULT	50
ADD	51	SUB	52	DEC	53	DIV	54
NUML.	55	SCROLL	56	LWIN	57	RWIN	58
APPS	59	OEM1	60	PLUS	61	COMMA	62
MINUS	63	PERIOD	64	OEM2	65	OEM3	66
OEM2	67	OEM5	68	OEM6	69	OEM7	70

The OEM1-7 keys have different meanins depending on the type of keyboard. US and NO mappings are given in the following table:

KEY	US	NO	KEY	US	NO
OEM1	; :	¨ ^	OEM2	/ ?	' *
OEM3	` ~	∅	OEM4	[{	\ `
OEM5	\ |	| §	OEM6	] }	Å
OEM7	' ″	Æ			

1.1.11 LoadSensor

Loadsensor {

exposedField	SFBool	**enabled**	TRUE	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFTime	**timeOut**	0	
exposedField	MFNode	**watchList**	[]	
field	SFBool	**watchBrowser**	FALSE	(OctagaSpecific)
eventOut	SFBool	**isActive**		
eventOut	SFBool	**isLoaded**		

eventOut	SFTime	**loadTime**		
eventOut	SFFloat	**progress**		

}

The **Loadsensor** node is identical to the standard X3D **LoadSensor** except for one extra field. The **watchBrowser** field makes the LoadSensor report the progress of the next file being loaded by the player instead of the **watchList.** This feature is used by the Octaga loader screen.

1.1.12 MovieTexture

MovieTexture {

exposedField	SFBool	**loop**	FALSE	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFTime	**pauseTime**	0	
exposedField	SFTime	**resumeTime**	0	
exposedField	SFFloat	**speed**	1.0	
exposedField	SFTime	**startTime**	0	
exposedField	SFTime	**stopTime**	0	
exposedField	MFString	**url**	[]	(Octaga Specific)
field	SFBool	**fastResize**	TRUE	**(deprecated)**
field	SFBool	**repeatS**	TRUE	
field	SFBool	**repeatT**	TRUE	
field	SFNode	**textureProperties**	NULL	
eventOut	SFTime	**duration_changed**		
eventOut	SFTime	**elapsedTime**		
eventOut	SFBool	**isActive**		
eventOut	SFBool	**isPaused**		
eventOut	SFTime	**playbackFraction**		(Octaga Specific)

}

The **MovieTexture** node is identical to the standard X3D **MovieTexture** except for one extra field. The **fastResize** field enables fast resizing of the video in order to fit a texture at the expense of not being able to tile the texture. This can be used to speed ut the rendering of non-power-of-two textures that are not to be tiled. This field has been deprectated as most modern GPUs can handle non-power-of-two textures seamlessly. In addition the **url** field can have one special value "*live" setting the source of the movitexture to the first available video capture device detected. The **playbackFraction** returns the current position of the video playback (from 0 to 1) as this might be slightly out of sync with the internal timer.

1.1.13 MultiTexture

MultiTexture {

exposedField	SFFloat	**alpha**	1
exposedField	SFColor	**color**	1 1 1
exposedField	MFString	**function**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	MFString	**mode**	[]
exposedField	MFString	**source**	[]
exposedField	MFNode	**texture**	[]

}

The MultiTexture node is identical to the standard X3D MultiTexture node except for one extra legal values for the **source** field. In addition to the standard values Octaga accepts "**WHITE**" signifying that the texture should be lit but that the diffusecolors specified in the Material node should be ignored.

1.1.14 NavigationInfo

NaviagationInfo {

eventIn	SFBool	**set_bind**		
exposedField	SFNode	**avatar**	NULL	(Octaga Specific)
exposedField	MFFloat	**avatarSize**	[0.25, 1.6, 0.75]	
exposedField	SFVec3f	**enableRightClickMenu**	TRUE	(Octaga Specific)
exposedField	SFVec3f	**gravity**	0, -9.8, 0	(Octaga Specific)
exposedField	SFBool	**headlight**	TRUE	
exposedField	SFNode	**metadata**	NULL	
exposedField	MFString	**navigationDevice**	[]	(Octaga Specific)
exposedField	SFFloat	**rotationSpeed**	1.0	(Octaga Specific)
exposedField	SFFloat	**speed**	1.0	
exposedField	MFFloat	**transitionTime**	1.0	
exposedField	MFString	**transitionType**	“LINEAR”	
exposedField	MFString	**type**	[“EXAMINE”,”ANY”]	
exposedField	SFFloat	**visibilityLimit**	0.0	
eventOut	SFTime	**bindTime**		
eventOut	SFBool	**isBound**		
eventOut	SFBool	**transitionComplete**		

}

The **NavigationInfo** node is identical to the standard X3D **NavigationInfo** node except for five extra fields.
The **avatar** field allows you to put in an avatarNode node in the scene graph.
The ***enableRightClickMenu*** field controls whether or not the rightClicMenu should be enabled or not. This can be useful if you want to use the right click for something else using a MouseSensor.
The **gravity** field allows specifying the direction and magnitude of the gravity. This also implies the upVector when navigating.
The **navigationDevice** field controls which devices are used for navigation. Valid strings is on the form "XXX:YYY" where XXX is either ENABLE or DISABLE and YYY is ALL,MOUSE,KEYBOARD or JOYSTICK (currently). The MF values are processed sequentially, thus a value of: ["ENABLE:ALL" , "DISABLE:MOUSE"] will enable all devices except the mouse, and ["DISABLE:ALL","ENABLE:JOYSTICK","ENABLE:KEYBOARD"] will disable all devices except the joystick and keyboard.
The ***rotationSpeed*** field specifies the rate at which the viewer rotates in a scene.

1.1.15 ParticleSystem

ParticleSystem {

exposedField	SFNode	**appearance**	NULL	
exposedField	SFBool	**createParticles**	TRUE	
exposedField	SFNode	**geometry**	NULL	
exposedField	SFBool	**enabled**	TRUE	
exposedField	SFFloat	**lifetimeVariation**	0.25	
exposedField	SFInt32	**maxParticles**	200	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFFloat	**particleLifetime**	5	
exposedField	SFVec2f	**particleSize**	0.02 0.02	
exposedField	SFVec2f	**particleSizeVariation**	0 0	(Octaga Specific)
exposedField	SFFloat	**creationRate**	0	(Octaga Specific)
exposedField	SFFloat	**creationRateVariation**	0	(Octaga Specific)
eventOut	SFBool	**isActive**		
field	SFVec3f	**bboxCenter**	0 0 0	
field	SFVec3f	**bboxSize**	-1 -1 -1	
field	SFNode	**colorRamp**	NULL	

field	MFFloat	**colorKey**	[]		
field	MFFloat	**colorVariationHSV**	[]		(Octaga Specific)
field	SFNode	**emitter**	NULL		
field	SFString	**geometryType**	"QUAD"		
field	MFNode	**physics**	[]		
field	MFVec3f	**scaleRamp**	[]		(Octaga Specific)
field	MFFloat	**scaleKey**	[]		(Octaga Specific)
field	SFNode	**texCoordRamp**	NULL		
field	MFFloat	**texCoordKey**	[]		
field	SFVec3f	**pointAttenuation**	1 0 0		(Octaga Specific)
field	SFFloat	**pointFadeThreshold**	1		(Octaga Specific)

}

The ParticleSystem node is identical to the standard X3D ParticleSystem node except for some additional fields. The **particleSizeVariation** field allows particles have a variation in size when beeing emmitted. The **creationRate** and **creationRateVariation** offers additional control over how fast new particles are created. If set to zero these two fields are ignored.The colorVariationHSV field contains triplets of HSV colors (each component in the range [0,1]) defining the color variation for each color in the **colorRamp**. The **scaleRamp** and **scaleKey** defines how a particle is scaled over the lifetime of an individual particle. The **scaleKey** represents the age of the particle in seconds while the **scaleRamp** defines the scale at these key points. Between keys the values are scaled linearly. A scale value of 1 1 1 means no scaling. The **pointAttenuation** and **pointFadeThreshold** give addition control over the rendering of points and sprites and works as described in the PointProperties node.

1.1.16 PointEmitter

PointEmitter {

exposedField	SFVec3f	**direction**	0 1 0		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFVec3f	**position**	0 0 0		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**directionVariation**	0	[0-inf]	(Octaga Specific)

}

The PointEmitter node is identical to the standard X3D PointEmitter except for four additional fields. The speedVariation, massVariation, surfaceAreaVarion and directionVariation allows the variation for speed, mass, surfaceArea and direction to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.1.17 PointSet

PointSet {

exposedField	MFNode	**attrib**	NULL	
exposedField	SFNode	**color**	NULL	
exposedField	SFNode	**coord**	NULL	
exposedField	SFNode	**fogCoord**	NULL	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFNode	**pointProperties**	NULL	(Octaga Specific)

}

The PointSet node is identical to the standard X3D PointSet except for one extra field. The **pointProperties** field allows to add a PointProperties node to specify the appearance of the individual points.

1.1.18 PolylineEmitter

PolylineEmitter {

eventIn	SFInt32	**set_coordinate**			
exposedField	SFNode	**coords**	NULL		
exposedField	SFVec3f	**direction**	0 1 0		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
exposedField	MFInt32	**coordIndex**	[]		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**directionVariation**	0	[0-inf]	(Octaga Specific)

}

The PolylineEmitter node is identical to the standard X3D PolylineEmitter except for four additional fields. The speedVariation, massVariation, surfaceAreaVarion and directionVariation allows the variation for speed, mass, surfaceArea and direction to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.1.19 ProximitySensor

ProximitySensor {

exposedField	SFVec3f	**center**	0 0 0	
exposedField	SFBool	**enabled**	TRUE	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFVec3f	**size**	0 0 0	
exposedField	SFBool	**alwaysOutput**	FALSE	(Octaga Specific)
eventOut	SFTime	**enterTime**		
eventOut	SFTime	**exitTime**		
eventOut	SFVec3f	**centerOfRotation_changed**		
eventOut	SFBool	**isActive**		
eventOut	SFRotation	o**rientation_changed**		
eventOut	SFVec3f	**position_changed**		

}

The ProximitySensor node is identical to the standard X3D ProximitySensor except for one extra field. The **alwaysOutput** field can be used to enable the output of event also during viewpoint transitions triggered by the user. This can be used to avoid jumpy HUDs.

1.1.20 RigidBodyCollection

RigidBodyCollection {

eventIn	MFNode	**set_contacts**	[]	
exposedField	SFFloat	**angularDamping**	0	(Octaga Specific)
exposedField	SFBool	**autoDisable**	FALSE	
exposedField	MFNode	**bodies**	[]	
exposedField	SFFloat	**constantForceMix**	0.0001	
exposedField	SFFloat	**contactSurfaceThickness**	0	
exposedField	SFFloat	**disableAngularSpeed**	0	
exposedField	SFFloat	**disableLinearSpeed**	0	
exposedField	SFFloat	**disableTime**	0	
exposedField	SFBool	**enabled**	0	
exposedField	SFFloat	**errorCorrection**	0.8	
exposedField	SFVec3f	**gravity**	0 -9.8 0	
exposedField	SFInt32	**iterations**	10	
exposedField	MFNode	**joints**	[]	
exposedField	SFFloat	**linearDamping**	0	(Octaga Specific)
exposedField	SFFloat	**maxCorrectionSpeed**	-1	
exposedField	SFNode	**metadata**	NULL	
exposedField	SFBool	**preferAccuracy**	FALSE	
field	SFNode	**collider**	NULL	

}

The ProximitySensor node is identical to the standard X3D ProximitySensor except for two extra field. The **angularDamping** field can be used to damp any angularMovement of the bodies connected to the RigidBodyCVollection, while the **linearDamping** field can be used to damp linearMovement.

1.1.21 ScreenFontStyle

ScreenFontStyle {

exposedField	SFNode	**metadata**	NULL	
field	MFString	**family**	[“SERIF”]	
field	SFBool	**horizontal**	TRUE	
field	MFString	**justify**	[“BEGIN”]	
field	SFString	**language**	“”	
field	SFBool	**leftToRight**	TRUE	
field	SFFloat	**pointSize**	1.0	
field	SFFloat	**spacing**	1.0	
field	SFString	**style**	“PLAIN”	
field	SFBool	**topToBottom**	TRUE	
field	SFBool	**cullable**	“”	(Octaga Specific)

}

The ScreenFontStyle node is identical to the standard X3D ScreeFontStyle node except for one additional fields. The **cullable** field specifies if the text can be culled by the renderer or not. This is nescesarry as the bounding box of the text using the screenFontStyle cannot be accurately calculated. If cullling is enabled a small box at the local origo is used as the bounding box. The size dependant output fields of the Text node is set as if it contained a regular FontStyle node with size set to one.

1.1.22 ShaderPart

ShaderPart {

exposedField	SFNode	**metadata**	NULL
exposedField	MFString	**url**	[]

field	SFString	**type**	“VERTEX”	
field	SFString	**geometryInputType**	“LINES”	(Octaga Specific)
field	SFString	**geometryOutputType**	“LINE_STRIP”	(Octaga Specific)
field	SFInt32	**geometryVerticesOut**	0	(Octaga Specific)

}

The ShaderPart node is identical to the standard X3D ShaderPart node except for three additional fields and one additional legalValue for the type field. The three new fields are ony used when the type field is set to "GEOMETRY". The **geometryInputType** field then specifies what type of geometry to use as input for the geometryShader. Legal values include "POINTS", "LINES", "LINES_ADJACENCY", "TRIANGLES" and "TRIANGLES_ADJACENCY". The **geometryOutputType** field similarely specifies what type of geometry to emit from the shader. Legal values include "POINTS", "LINE_STRIP" and "TRIANGLES_STRIP". The **geometryVericesOut** field specifies the maximum number of vertices emmitted by the shader in one pass. If this value is 0 (the default) the maximum allowed with the given hardware.

1.1.23 ShaderProgram

ShaderProgram {

exposedField	SFNode	**metadata**	NULL	
exposedField	MFString	**url**	[]	
field	SFString	**type**	“VERTEX”	
field	SFString	**geometryInputType**	“LINES”	(Octaga Specific)
field	SFString	**geometryOutputType**	“LINE_STRIP”	(Octaga Specific)
field	SFInt32	**geometryVerticesOut**	0	(Octaga Specific)
# And any number of				
eventIn	fieldType	**fieldName**		
exposedField	fieldType	**fieldName**	initialValue	
eventOut	fieldType	**fieldName**		
field	fieldType	**fieldName**	initialValue	

}

The ShaderProgram node is identical to the standard X3D ShaderProgram node except for three additional fields and one additional legalValue for the type field. The three new fields are ony used when the type field is set to "GEOMETRY". The **geometryInputType** field then specifies what type of geometry to use as input for the geometryShader. Legal values include "POINTS", "LINES", "LINES_ADJACENCY", "TRIANGLES" and "TRIANGLES_ADJACENCY". The **geometryOutputType** field similarely specifies what type of geometry to emit from the shader. Legal values include "POINTS", "LINE_STRIP" and "TRIANGLES_STRIP". The **geometryVericesOut** field specifies the maximum number of vertices emmitted by the shader in one pass. If this value is 0 (the default) the maximum allowed with the given hardware.

1.1.24 StaticGroup

StaticGroup {

exposedField	SFNode	**metadata**	NULL	
field	SFBool	**useDisplayList**	FALSE	(Octaga Specific)
field	MFNode	**children**	[]	
field	SFVec3f	**bboxCenter**	0 0 0	
field	SFVec3f	**bboxSize**	-1 -1 -1	

}

The StaticGroup node is identical to the standard X3D StaticGroup except for one additional field. The **useDisplayList** field can be used to put all the geometry of a StaticGroup into an OpenGL display list. This can improve the performance in some applications, butmay have undesired sede effects. Use with caution.

1.1.25 SquadOrientationInterpolator

SquadOrientationInterpolator {

eventIn	SFFloat	**set_fraction**			
exposedField	SFBool	**closed**	FALSE		(Octaga Specific)
exposedField	MFFloat	**key**	[]		
exposedField	MFRotation	**keyValue**	[]		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFBool	**normalizeVelocity**	FALSE		
eventOut	SFRotation	**value_changed**			

}

The **SquadOrientationInterpolator** node is identical to the standard X3D **SquadOrientationInterpolator** except for one additional field. The **closed** field specifies whether the interpolator should provide a closed loop, with continuous velocity vectors as the interpolator transitions from the last key to the first key. If the velocity vectors at the first and last keys are specified, the closed field is ignored. If the keyValues at the first and last key are not identical, the closed field is ignored.

1.1.26 SurfaceEmitter

PolylineEmitter {

eventIn	SFInt32	**set_coordinate**			
exposedField	SFNode	**coords**	NULL		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
exposedField	MFInt32	**coordIndex**	[]		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)

}

The PolylineEmitter node is identical to the standard X3D PolylineEmitter except for three additional fields. The speedVariation, massVariation and surfaceAreaVarion allows the variation for speed, mass and surfaceArea to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.1.27 TextureBackground

TextureBackground {

eventIn	SFBool	**set_bind**		
exposedField	SFNode	**metadata**	NULL	
exposedField	MFFloat	**groundAngle**	[]	
exposedField	MFColor	**groundColor**	[]	
exposedField	SFNode	**backTexture**	NULL	
exposedField	SFNode	**backTextureTransform**	NULL	(Octaga Specific)
exposedField	SFNode	**bottomTexture**	NULL	
exposedField	SFNode	**bottomTextureTransform**	NULL	(Octaga Specific)
exposedField	SFNode	**frontTexture**	NULL	
exposedField	SFNode	**frontTextureTransform**	NULL	(Octaga Specific)
exposedField	SFNode	**leftTexture**	NULL	
exposedField	SFNode	**leftTextureTransform**	NULL	(Octaga Specific)
exposedField	SFNode	**rightTexture**	NULL	
exposedField	SFNode	**rightTexture Transform**	NULL	(Octaga Specific)

exposedField	SFNode	**topTexture**	NULL	
exposedField	SFNode	**topTextureTransform**	NULL	(Octaga Specific)
exposedField	MFFloat	**skyAngle**	[]	
exposedField	MFColor	**skyColor**	0 0 0	
exposedField	SFFloat	**transparency**	0	
eventOut	SFTime	**bindTime**		
eventOut	SFBool	**isBound**		

}

The TextureBackground node is identical to the standard X3D TextureBackground except that each of the six texture fields has a corresponding textureTransform field, These fields can be useful if you need to flip, tile, scale or rotate the textures used in the TextureBackground.

1.1.28 TextureProperties

TextureProperties {

exposedField	SFFloat	**anisotropicDegree**	1.0	
exposedField	SFColorRGBA	**borderColor**	0 0 0 0	
exposedField	SFInt32	**borderWidth**	0	
exposedField	SFString	**boundaryModeS**	"REPEAT"	
exposedField	SFString	**boundaryModeT**	"REPEAT"	
exposedField	SFString	**boundaryModeR**	"REPEAT"	
exposedField	SFString	**magnificationFilter**	"FASTEST"	(Extra Values)
exposedField	SFNode	**metadata**	NULL	
exposedField	SFString	**minificationFilter**	"FASTEST"	(Extra Values)
exposedField	SFString	**textureCompression**	"FASTEST"	(Extra Values)
exposedField	SFFloat	**texturePriority**	0	
field	SFString	**format**	“”	(Octaga Specific)
field	SFBool	**generateMipMaps**	FALSE	

}

The TextureProperties node is identical to the standard X3D TextureProperties node except for one additional field. The format field defines the desired internal format of the texture according to the table below. The conversion from the default format for the given texture may not be possible, in this case the default format is used.

MagnificationFilter	**OpenGL equivalent**
“NICEST” “AVG_PIXEL” “LINEAR”	LINEAR
“FASTEST” “NEAREST_PIXEL” “NEAREST”	NEAREST
“DEFAULT”	Use application settings

MinificationFilter	**OpenGL equivalent**
“AVG_PIXEL” “LINEAR”	LINEAR
“NICEST” “AVG_PIXEL_AVG_MIPMAP” “LINEAR_MIPMAP_LINEAR”	LINEAR_MIPMAP_LINEAR

“AVG_PIXEL_NEAREST_MIPMAP” “LINEAR_MIPMAP_NEAREST”	LINEAR_MIPMAP_NEAREST
“FASTEST” “NEAREST_PIXEL” “NEAREST”	NEAREST
“NEAREST_PIXEL_AVG_MIPMAP” “NEAREST_MIPMAP_LINEAR”	NEAREST_MIPMAP_LINEAR
“NEAREST_PIXEL_NEAREST_MIPMAP” “NEAREST_MIPMAP_NEAREST”	NEAREST_MIPMAP_NEAREST
“DEFAULT”	Use application settings

TextureCompression	**OpenGL equivalent**
“OFF” “LOW” “NICEST”	Texture compression off
“ON” “FASTEST” “HIGH” “MEDIUM”	Texture compression on. The type of compression depends on the texture data and the hardware capabilities.
“DEFAULT”	Use application settings

Format	**Description**
“A”	Alpha, 8 bits per pixel
“L”	Luminance, 8 bits per pixel
“L16”	Luminance, 16 bits per pixel
“LA”	Luminance, Alpha, 8 bits per component (16 bits per pixel)
“RGB”	RGB, 8 bits per component (24 bits per pixel)
“BGR”	BGR, 8 bits per component (24 bits per pixel)
“R3G3B2”	RGB, 8 bits per pixel
“R5G6B5”	RGB, 16 bits per pixel
"X1R5G5B5"	RGB, 16 bits per pixel
"X4R4G4B4"	RGB, 16 bits per pixel
"XRGB"	RGB, 8 bits per component (32 bits per pixel)
"XBGR"	BGR, 8 bits per component (32 bits per pixel)
"RGBA"	RGBA, 8 bits per component (32 bits per pixel)
"ARGB"	ARGB, 8 bits per component (32 bits per pixel)
"ABGR"	ABGR, 8 bits per component (32 bits per pixel)
"A1R5G5B5"	ARGB, 16 bits per pixel
"A4R4G4B4"	ARGB, 16 bits per pixel
"A2R10G10B10"	ARGB, 32 bits per pixel

"A2B10G10R10"	ABGR, 32 bits per pixel
"A16B16G16R16"	ABGR, 64 bits per pixel
"L16F"	Luminance, floating point values, 16 bits per pixel
"L32F"	Luminance, floating point values, 32 bits per pixel
"L16A16F"	Luminance, Alpha, floating point values, 16 bits per component (32 bits per pixel)
"L32A32F"	Luminance, Alpha, floating point values, 32 bits per component (64 bits per pixel)
"RGBA16F"	RGBA, floating point values, 16 bits per component (64 bits per pixel)
"RGBA32F"	RGBA, floating point values, 32 bits per component (128 bits per pixel)
"RGB_DXT1"	RGB, Compressed using DXT1
"RGBA_DXT3"	RGBA, Compressed using DXT3
"RGBA_DXT5"	RGBA, Compressed using DXT5

1.1.29 TriangleSet

TriangleSet {

exposedField	MFNode	**attrib**	[]
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**fogCoord**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	SFNode	**texCoord**	NULL
field	SFBool	**ccw**	TRUE
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**normalPerVertex**	TRUE
field	SFBool	**solid**	TRUE
field	SFBool	**static**	FALSE **(deprecated)**

}

The TriangleSet node is identical to the standard X3D TriangleSet node except for one additional field. The static field optimizes the rendering of a static geometry and can lead to substansial performance improvements on modern GPUs.

1.1.30 TriangleStripSet

TriangleStripSet {

exposedField	MFNode	**attrib**	[]
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**fogCoord**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	SFNode	**normal**	NULL
exposedField	MFInt32	**stripCount**	[]
exposedField	SFNode	**texCoord**	NULL
field	SFBool	**ccw**	TRUE
field	SFBool	**colorPerVertex**	TRUE

field	SFBool	**normalPerVertex**	TRUE	
field	SFBool	**solid**	TRUE	
field	SFBool	**static**	FALSE	**(deprecated)**

}

The TriangleStripSet node is identical to the standard X3D TriangleStripSet node except for one additional field. The static field optimizes the rendering of a static geometry and can lead to substansial performance improvements on modern GPUs.

1.1.31 VolumeEmitter

PolylineEmitter {

eventIn	SFInt32	**set_coordinate**			
exposedField	SFNode	**coords**	NULL		
exposedField	SFVec3f	**direction**	0 1 0		
exposedField	SFNode	**metadata**	NULL		
exposedField	SFFloat	**speed**	0		
exposedField	SFFloat	**variation**	0.25		
exposedField	MFInt32	**coordIndex**	[]		
field	SFBool	**internal**	TRUE		
field	SFFloat	**mass**	0		
field	SFFloat	**surfaceArea**	0		
exposedField	SFFloat	**speedVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**massVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**surfaceAreaVariation**	0	[0-inf]	(Octaga Specific)
exposedField	SFFloat	**directionVariation**	0	[0-inf]	(Octaga Specific)

}

The VolumeEmitter node is identical to the standard X3D VolumeEmitter except for four additional fields. The speedVariation, massVariation, surfaceAreaVarion and directionVariation allows the variation for speed, mass, surfaceArea and direction to be specified independantly. If one of the specific variation fields is set to zero. The global variation field is used instead.

1.2 MPEG-4 nodes

The following MPEG-4 nodes are implemented:

1.2.1 Background2D

Background2D {

eventIn	SFBool	**set_bind**	
exposedField	SFColor	**backColor**	0 0 0
exposedField	SFNode	**metadata**	NULL
exposedField	MFString	**url**	[]
eventOut	SFBool	**isBound**	

}

There exists a **Background2D** stack, in which the top-most background is the current active background one. The **Background2D** node allows a background to be displayed behind a 2D scene. The functionality of this node can also be accomplished using other nodes, but use of this node may be more efficient in some implementations.

If **set_bind** is set to TRUE the **Background2D** is moved to the top of the stack.If **set_bind** is set to FALSE, the **Background2D** is removed from the stack so the previous background which is contained in the stack is on top again.

The **isBound** event is sent as soon as the backdrop is put at the top of the stack, so becoming the current backdrop.

The **url** field specifies the data source to be used

The **backColor** field specifies a colour to be used as the background.

This is not a geometry node. The top-left corner of the image is mapped to the top-left corner of the **Layer2D** and the right-bottom corner of the image is stretched to the right-bottom corner of the **Layer2D**, regardless of the current transformation. Scaling and/or rotation do not have any effect on this node. The background image will always exactly fill the entire **Layer2D**, regardless of **Layer2D** size, without tiling or cropping.

When a **Background2D** node is included in a 3D context, that is in a **Group**, **Layer3D**, or **CompositeTexture3D** node, then it shall be rendered behind all other geometries and be scaled to fit in the enclosing frame. For Group node, this frame is the whole scene. For **Layer3D** and **CompositeTexture3D** the background image is scaled to fit in the frame of the node.

1.2.2 Bitmap

Bitmap {

exposedField	SFNode	**metadata**	NULL
exposedField	SFVec2f	**scale**	-1, -1

}

Bitmap is a geometry node centered at (0,0) in the local coordinate system, to be placed in the geometry field of a **Shape** node. It is a screen-aligned rectangle, which means that the surface normal of this rectangle will always be in the same direction as the screen surface normal, namely straight out to the viewer. It is for example not possible to view the **Bitmap** under an angle from the side. **Bitmap** has the dimensions of the texture that is mapped onto it, as specified in the **Appearance** node of its parent **Shape** node. However, the effective geometry of **Bitmap** is defined by the non-transparent pixels of the image or video that is mapped onto it.

When no scaling is specified, a trivial texture-mapping (pixel copying) is performed.
The **scale** field specifies a scaling of the geometry in the x and y dimensions, respectively. The **scale** values shall be strictly positive or equal to -1. A **scale** value of -1 indicates that no scaling shall be applied in the relevant dimension. The special case where both scale dimensions are -1 indicates that the natural dimensions of the texture that is mapped onto the **Bitmap** shall be used. **Bitmap** shall not be rotated but may be subject to translation.

Geometry sensors shall respond to the effective geometry of the **Bitmap**, which is defined by the non-transparent pixels of the texture that is mapped onto it.

If a **Material** or **Material2D** node is specified in the **appearance** of the parent **Shape** of a **Bitmap**, the final transparency of each pixel is given by the **Material** or **Material2D** transparency multiplied by the transparency (1-alpha) value of each pixel of the texture. If the texture has no alpha plane then the final transparency of each pixel is purely given by **Material** or **Material2D** transparency (as if the texture had an alpha value of 0).

1.2.3 Circle

Deprecated: Use Circle2D or Disk2D instead

Circle {

exposedField	SFNode	**metadata**	NULL

exposedField	SFFloat	**radius**	1.0

}

This node specifies a circle centered at (0,0) in the local coordinate system. The **radius** field specifies the radius of the circle and shall be greater than 0. The default texture mapping coordinates are defined as the four corners of the bounding box of the circle.

1.2.4 CompositeTexture2D

CompositeTexture2D {

eventIn	MFNode	**addChildren**		
eventIn	MFNode	**removeChildren**		
exposedField	SFNode	**background**	NULL	
exposedField	MFNode	**children**	[]	
exposedField	SFBool	**isPickable**	TRUE	(Octaga Specific)
exposedField	SFNode	**metadata**	NULL	(Octaga Specific)
exposedField	SFInt32	**pixelWidth**	-1	
exposedField	SFInt32	**pixelHeight**	-1	
exposedField	SFInt32	**samples**	**0**	(Octaga Specific)
exposedField	MFNode	**textureProperties**	[]	(Octaga Specific)
exposedField	SFString	**update**	“ALWAYS”	(Octaga Specific)
exposedField	SFNode	**viewport**	NULL	

}

The **CompositeTexture2D** node represents a texture that is composed of a 2D scene, which may be mapped onto another object.

This node may only be used as the texture field of an **Appearance** node. All behaviors and user interaction are enabled when using a **CompositeTexture2D**.

The **addChildren** eventIn specifies a list of nodes that shall be added to the **children** field.

The **removeChildren** eventIn specifies a list of nodes that shall be removed from the **children** field.

The **children** field contains a list of 2D children nodes that define the 2D scene that is to form the texture map.

The **pixelWidth** and **pixelHeight** fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The semantics of the **background** and **viewport** fields are identical to the semantics of the **Layer2D** fields of the same name.

Octaga Specific:
The **isPickable** field defines if the children nodes takes part in picking operations.

The **metadata** field allows metadata to be attched to the node.

The **samples** field allows the generated texture to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

textureProperties allows fine control over a texture's application. Note that the for a CompositeTexture2D only the following values are valid for the **format** field of the TextureProperties nodes:

Format	Description
“RGBA”	RGBA, 8 bits per component (32 bits per pixel)
“RGBA32F”	RGBA, floating point values, 32 bits per component (128 bits per pixel)
“DEPTH”	DEPTH format

If specifying more than one TextureProperty node Octaga will create multiple output buffers. Theese buffers can be adressed by shaders inside the children field. To get Access to the differnt output buffers use the RenderBuffer node.

The **update** field allows the user to request a regeneration of the texture. Setting this field to "ALWAYS" will cause the texture to be rendered every frame. A value of "NONE" will stop rendering so that no further updates are performed even if the contained scene graph changes. When the value is set to "NEXT_FRAME_ONLY", it is an instruction to render the texture at the end of this frame, and then not render it again. In this case, the update frame indicator is set to this frame; at the start of the next frame, the update value shall be automatically set back to "NONE" to indicate that the rendering has already taken place. Since this is a change of value for the **update** field, an output event is automatically generated.

1.2.5 CompositeTexture3D

CompositeTexture3D {

eventIn	MFNode	**addChildren**		
eventIn	MFNode	**removeChildren**		
exposedField	MFNode	**children**	[]	
exposedField	SFInt32	**pixelWidth**	-1	
exposedField	SFInt32	**pixelHeight**	-1	
exposedField	SFNode	**background**	NULL	
exposedField	SFNode	**fog**	NULL	
exposedField	SFNode	**navigationInfo**	NULL	
exposedField	SFNode	**viewpoint**	NULL	
exposedField	SFString	**update**	“ALWAYS”	(Octaga Specific)
exposedField	SFNode	**metadata**	NULL	(Octaga Specific)
exposedField	MFNode	**textureProperties**	[]	(Octaga Specific)
exposedField	SFBool	**isPickable**	TRUE	(Octaga Specific)
exposedField	SFInt32	**samples**	**0**	(Octaga Specific)

}

The **CompositeTexture3D** node represents a texture mapped onto a 3D object that is composed of a 3D scene.

Behaviors and user interaction are enabled when using a **CompositeTexture3D**. However, the standard user navigation on the textured scene is disabled. Instead, sensors contained in the scene which forms the **CompositeTexture3D** may be used to define behaviours. This node may only be used as a **texture** field of an **Appearance** node.

The **addChildren** eventIn specifies a list of nodes that shall be added to the **children** field.

The **removeChildren** eventIn specifies a list of nodes that shall be removed from the **children** field.

The **children** field is the list of 3D children nodes that define the 3D scene that forms the texture map.

The **pixelWidth** and **pixelHeight** fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The **background**, **fog**, **navigationInfo** and **viewpoint** fields represent the current values of the bindable children nodes used in the 3D scene.

Octaga Specific:
The **update** field allows the user to request a regeneration of the texture. Setting this field to "ALWAYS" will cause the texture to be rendered every frame. A value of "NONE" will stop rendering so that no further updates are performed even if the contained scene graph changes. When the value is set to "NEXT_FRAME_ONLY", it is an instruction to render the texture at the end of this frame, and then not render it again. In this case, the update frame indicator is set to this frame; at the start of the next frame, the update value shall be automatically set back to "NONE" to indicate that the rendering has already taken place. Since this is a change of value for the **update** field, an output event is automatically generated.

The **metadata** field allows metadata to be attched to the node.

The **samples** field allows the generated texture to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

textureProperties allows fine control over a texture's application. Note that the for a CompositeTexture3D only the following values are valid for the **format** field of the TextureProperties nodes:

Format	**Description**
“RGBA”	RGBA, 8 bits per component (32 bits per pixel)
“RGBA32F”	RGBA, floating point values, 32 bits per component (128 bits per pixel)
“DEPTH”	DEPTH format

If specifying more than one TextureProperty node Octaga will create multiple output buffers. Theese buffers can be adressed by shaders inside the children field. To get Access to the differnt output buffers use the RenderBuffer node.

The isPickable field defines if the children nodes takes part in picking operations. To avoid navigation inside the CompositeTexture3D use a navigationInfo with navigation type set to "NONE".

1.2.6 Coordinate2D

Coordinate2D {
exposedField SFNode **metadata** NULL
exposedField MFVec2f **point** []
}

This node defines a set of 2D coordinates to be used in the **coord** field of geometry nodes. The **point** field contains a list of points in the 2D coordinate space.

1.2.7 DiscSensor

DiscSensor {
exposedField SFBool **autoOffset** TRUE

exposedField	SFBool	**enabled**	TRUE
exposedField	SFFloat	**maxAngle**	-1.0
exposedField	SFFloat	**minAngle**	0.0
exposedField	SFNode	**metadata**	NULL
exposedField	SFFloat	**offset**	0.0
EventOut	SFBool	**isActive**	
EventOut	SFFloat	**rotation_changed**	
EventOut	SFVec2f	**trackPoint_changed**	

}

This sensor enables the rotation of an object in the 2D plane around an axis specified in the local coordinate system. The semantics are as similar to those for **CylinderSensor**, but restricted to a 2D case.

1.2.8 Form

Form {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	SFVec2f	**size**	-1, -1
exposedField	MFInt32	**groups**	[]
exposedField	MFInt32	**constraints**	[]
exposedField	MFInt32	**groupsIndex**	[]

}

The **Form** node specifies the placement of its children according to relative alignment and distribution constraints.
Distribution spreads objects regularly, with an equal spacing between them.
The **children** field shall specify a list of nodes that are to be arranged. The children's position is implicit and order
is important.
The **size** field specifies the width and height of the layout frame.
The **groups** field specifies the list of groups of objects on which the constraints can be applied. The children of the
Form node are numbered from 1 to n, 0 being reserved for a reference to the form itself. A group is a list of child
indices, terminated by a -1.
The **constraints** and the **groupsIndex** fields specify the list of constraints. One constraint is constituted by a
constraint type from the **constraints** field, coupled with a set of group indices terminated by a –1 contained in the
groupsIndex field. There shall be as many strings in **constraints** as there are –1-terminated sets in
groupsIndex. The n-th constraint string shall be applied to the n-th set in the **groupsIndex** field.
Constraints belong to two categories: alignment and distribution constraints.
Components referred to in the tables below are components whose indices appear in the list following the
constraint type. When rank is mentioned, it refers to the rank in that list.
The semantics of the **<s>**, when present in the name of a constraint, is the following. It shall be a number, integer
when the scene uses pixel metrics, and float otherwise, which specifies the space mentioned in the semantics of
the constraint.
Alignment Constraints

Alignment Constraints Type	Index	Effect
Align Left edges	“AL”	The xmin of constrained components becomes equal to the xmin of the left-most component.
Align centers Horizontally	“AH”	The (xmin+xmax)/2 of constrained components becomes equal to the (xmin+xmax)/2 of the group of constrained components as computed before this constraint is applied.
Align Right edges	“AR”	The xmax of constrained components becomes equal to the xmax of the right-most component.
Align Top edges	“AT”	The ymax of all constrained components becomes equal to the ymax of the top-most component.
Align centers Vertically	“AV”	The (ymin+ymax)/2 of constrained components becomes equal to the (ymin+ymax)/2 of the group of constrained components as computed before this constraint is applied.
Align Bottom edges	“AB”	The ymin of constrained components becomes equal to the ymin of the bottom-most component.
Align Left edges by specified space	“AL <s>”	The xmin of the second and following components become equal to the xmin of the first component plus the specified space.
Align Right edges by specified space	“AR <s>”	The xmax of the second and following components becomes equal to the xmax of the first component minus the specified space.
Align Top edges by specified space	“AT <s>”	The ymax of the second and following components becomes equal to the ymax of the first component minus the specified space.
Align Bottom edges by specified space	“AB <s>”	The ymin of the second and following components become equal to the ymin of the first component plus the specified space.

The purpose of distribution constraints is to specify the space between components, by making such pairwise gaps equal either to a given value or to the effect of filling available space.

Distribution Constraints

Distribution Constraints Type	Index	Effect
Spread Horizontally	“SH”	The differences between the xmin of each component and the xmax of the previous one all become equal. The first and the last component shall be constrained horizontally already.
Spread Horizontally in container	“SHin”	The differences between the xmin of each component and the xmax of the previous one all become equal. References are the edges of the layout.
Spread Horizontally by specified space	“SH <s>”	The difference between the xmin of each component and the xmax of the previous one all become equal to the specified space. The first component is not moved.

Spread Vertically	“SV”	The differences between the ymin of each component and the ymax of the previous one all become equal. The first and the last component shall be constrained vertically already.
Spread Vertically in container	“SVin”	The differences between the ymin of each component and the ymax of the previous one all become equal. References are the edges of the layout.
Spread Vertically by specified space	“SV <s>”	The difference between the ymin of each component and the ymax of the previous one all become equal to the specified space. The first component is not moved.

All objects start at the center of the **Form**. The constraints are then applied in sequence.

1.2.9 IndexedFaceSet2D

IndexedFaceSet2D {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
eventIn	MFInt32	**set_texCoordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**metadata**	NULL
exposedField	SFNode	**texCoord**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	SFBool	**convex**	TRUE
field	MFInt32	**coordIndex**	[]
field	MFInt32	**texCoordIndex**	[]

}

The **IndexedFaceSet2D** node is the 2D equivalent of the **IndexedFaceSet** node. The **IndexedFaceSet2D** node represents a 2D shape formed by constructing 2D faces (polygons) from 2D vertices (points) specified in the **coord** field. The **coord** field contains a **Coordinate2D** node that defines the 2D vertices, referenced by the **coordIndex** field. The faces of an **IndexedFaceSet2D** node shall not overlap each other.

The detailed semantics are identical to those for the **IndexedFaceSet** node, restricted to the 2D case, and with the additional differences described here.

If the **texCoord** field is NULL, a default texture coordinate mapping is calculated using the local 2D coordinate system bounding box of the 2D shape, as follows. The X dimension of the bounding box defines the S coordinates, and the Y dimension defines the T coordinates. The value of the S coordinate ranges from 0 to 1, from the left end of the bounding box to the right end. The value of the T coordinate ranges from 0 to 1, from the lower end of the bounding box to the top end.

When the Material2D indicates "filled" the faces (polygons) are drawn and each face (polygon) is filled on the insides according to the following simple inside rule:

To determine if a point is inside draw an imaginary line through the entire polygon and each time the line crosses the polygon’s border increment a counter that was initialized to zero. When the count is odd the line is inside, when the count is even the line is outside.

When **color** field is non-null the color(s) are used either to fill the faces or to draw outlines of the faces depending on whether **Material2D filled** field is true or false respectively. In addition, if the **filled** field is true and the **Material2D lineProps** field is non-null then lines are drawn using the **LineProperties lineColor**.
When **color** field is null then the faces are filled and outlines are drawn using the rules listed in the **Material2D** Node.

In all cases that outlines are drawn the lines are drawn using the **lineStyle** and **width** field values from the **Material2D lineProps**, whether explicitly specified, or default values when the field is null.

1.2.10 IndexedLineSet2D

IndexedLineSet2D {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**metadata**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	[]

}

The **IndexedLineSet2D** node specifies a collection of lines or polygons.

The **coord** field shall list the vertices of the lines. When **coordIndex** is empty, the order of vertices shall be assumed to be sequential in the **coord** field. Otherwise, the **coordIndex** field determines the ordering of the vertices, with an index of -1 representing an end to the current polyline.

If the **color** field is not NULL, it shall contain a **Color** node, and the colors are applied to the line(s) as with the **IndexedLineSet** node.

The lines shall be drawn using the **LineProperties** node (whether explicit or default) attributes of **lineStyle** and **width**. If the **IndexedLineSet2D color** field is null then the **Material2D** is used to set the color of all the lines and **emissiveColor** shall be used unless the **lineProps** field is non-null when the **LineProperties**
lineColor shall be used instead.

1.2.11 Layer2D

Layer2D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	NULL
exposedField	SFNode	**metadata**	NULL
exposedField	SFVec2f	**size**	-1, -1
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**viewport** NULL	

}

The **Layer2D** node is a transparent rendering rectangle region on the screen where a 2D scene is drawn. The rectangle always faces the viewer of the scene. **Layer2D** and **Layer3D** nodes enable the composition of multiple 2D and 3D scenes (see Figure 24).

The **addChildren** eventIn specifies a list of 2D nodes that shall be added to the **Layer2D's children** field.

The **removeChildren** eventIn specifies a list of 2D nodes that shall be removed from the **Layer2D's children**

The **children** field may contain any 2D children nodes that define a 2D scene. Layer nodes are considered to be 2D objects within the scene. The layering of the 2D and 3D layers is specified by any relevant transformations in the scene graph. The **Layer2D** node is composed with its center at the origin of the local coordinate system and shall not be present in 3D contexts.

The **size** parameter shall be a floating point number that expresses the width and height of the layer in the units of the local coordinate system. In case of a layer at the root of the hierarchy, the size is expressed in terms of the default 2D coordinate system. A size of -1 in either direction, means that the **Layer2D** node is not specified in size in that direction, and that the size is adjusted to the size of the parent layer, or the global rendering area dimension if the layer is on the top of the hierarchy. In the case where a 2D scene or object is shared between several **Layer2D** nodes, the behaviours are defined exactly as for objects that are multiply referenced using the DEF/USE mechanism. A sensor triggers an event whenever the sensor is triggered in any of the **Layer2D** in which it is contained. The behaviors triggered by the shared sensors as well as other behaviors that apply on objects shared between several layers apply on all layers containing these objects.

A **Layer2D** stores the stack of bindable children nodes that can affect the children scene of the layer. All relevant bindable children nodes have a corresponding exposedField in the **Layer2D** node. During presentation, these fields take the value of the currently bound bindable children node for the scene that is a child of the **Layer2D** node. Initially, the bound bindable children node is the corresponding field value of the **Layer2D** node if it is defined. If the field is undefined, the first bindable children node defined in the child scene will be bound. When the binding mechanism of the bindable children node is used (**set_bind** field set to TRUE), all the parent layers containing this node set the corresponding field to the current bound node value. It is therefore possible to share scenes across layers, and to have different bound nodes active, or to trigger a change of bindable children node for all layers containing a given bindable children node. For 2D scenes, the **background** field specifies the bound **Background2D** node. The **viewport** field is reserved for future extensions for 2D scenes.

All the 2D objects contained in a single **Layer2D** node form a single composed object. This composed object is considered by other elements of the scene to be a single object. In other words, if a **Layer2D** node, A, is the parent of two objects, B and C, layered one on top of the other, it will not be possible to insert a new object, D, between B and C unless D is added as a child of A.

Layers are transparent to user input if the background field is set to NULL. If the background field is specified, any transparent part of the background will also let user input through to lower layers.

1.2.12 Layer3D

Layer3D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	NULL
exposedField	SFNode	**metadata**	NULL

exposedField	SFVec2f	**size**	-1, -1
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**fog**	NULL
exposedField	SFNode	**navigationInfo**	NULL
exposedField	SFNode	**viewpoint**	NULL

}

The **Layer3D** node is a transparent, rectangular rendering region where a 3D scene is drawn. The **Layer3D** node may be composed in the same manner as any other 2D node. It represents a rectangular region on the screen facing the viewer. The basic **Layer3D** semantics are identical to those for **Layer2D** but with 3D (rather than 2D) children. In general, **Layer3D** nodes shall not be present in 3D co-ordinate systems. The permitted exception to this in when a **Layer3D** node is the "top" node that begins a 3D scene or context.

The following fields specify bindable children nodes for **Layer3D**:
- **background** for **Background** and **Background2D** nodes
- **fog** for **Fog** nodes
- **navigationInfo** for **NavigationInfo** nodes
- **viewpoint** for **Viewpoint** nodes

The **viewpoint** field can be used to allow the viewing of the same scene with several viewpoints.

NOTE — The rule for transparency to behaviors is also true for navigation in **Layer3D**. Authors should carefully design the various **Layer3D** nodes in a given scene to take account of navigation. Overlapping several **Layer3D** with navigation turned on may trigger strange navigation effects which are difficult to control by the user. Unless it is a feature of the content, navigation can be easily turned off using the **NavigationInfo** type field, or **Layer3D's** can be designed not to be superimposed.

1.2.13 LinePropertiesMP4

LinePropertiesMP4 {

exposedField	SFColor	**lineColor**	0, 0, 0	
exposedField	SFInt32	**lineStyle**	0	
exposedField	SFNode	**metadata**		NULL
exposedField	SFFloat	**width**	1.0	

}

The MPEG-4 LineProperties node has been renamed LinepropertiesMP4 to avoid name conflict with the LineProperties node defined by X3D

The **LineProperties** node specifies line parameters used in 2D and 3D rendering.

The **lineColor** field specifies the color with which to draw the lines and outlines of 2D geometries.

The **lineStyle** field shall contain the line style type to apply to lines. The allowed values are:

lineStyle description

lineStyle	Description
0	solid
1	dash
2	dot
3	dash-dot
4	dash-dash-dot
5	dash-dot-dot

The terminal shall draw each line style in a manner that is distiguishable from each other line style.

The **width** field determines the width, in the local coordinate system, of rendered lines. The width is not subject to the local transformation.

The cap and join style to be used are as follows. The wide lines should end with a square form flush with the end of the lines.

1.2.14 Material2D

Material2D {

exposedField	SFColor	**emissiveColor**	0.8, 0.8, 0.8
exposedField	SFBool	**filled**	FALSE
exposedField	SFNode	**lineProps**	NULL
exposedField	SFNode	**metadata**	NULL
exposedField	SFFloat	**transparency**	0.0

}

The **Material2D** node specifies the characteristics of a rendered 2D **Shape**. Material2D shall be used as the material field of an Appearance node in certain circumstances.

The **emissiveColor** field specifies the color of the 2D **Shape**. If the shape is not filled, the interior is not drawn.

The **filled** field specifies whether rendered nodes are filled or drawn using lines. This field affects **IndexedFaceSet2D**, **Circle** and **Rectangle** nodes. If the rendered node is not filled the line shall be drawn centered on the rendered node outline. That means that half the line will fall inside the rendered node, and the other half outside.

The **lineProps** field contains information about line rendering in the form of a **LineProperties** node. When **filled** is true, if **lineProps** is null, no outline is drawn; if **lineProps** is non-null, an outline is drawn using **lineProps** information. When **filled** is false and **lineProps** is null, an outline is drawn with default width (1), default style (solid) and as line color the emissive color of the Material2D. When **filled** is false and **lineProps** is defined, line color, width and style are taken from the **lineProps** node. See also **LineProperties**.

The **transparency** field specifies the transparency of the 2D **Shape** and applies both to the filled interior as well as to the outline when drawn.

The part of the line which lies outside of the geometry shall not be sensitive to pointer activity.

When mapping texture onto a geometry and an outline is to be drawn, the texture shall first mapped onto the geometry, where the geometry dimensions are those without an outline. Then after the geometry is textured the outline shall be drawn.

1.2.15 OrderedGroup

OrderedGroup {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	MFFloat	**order**	[]

}

The **OrderedGroup** node controls the visual layering order of its children. When used as a child of a **Layer2D** node, it allows the control of which shapes obscure others. When used as a child of a **Layer3D** node, it allows content creators to specify the rendering order of elements of the scene that have identical z values. This allows conflicts between coplanar or close polygons to be resolved.

The **addChildren** eventIn specifies a list of objects that shall be added to the **OrderedGroup** node.

The **removeChildren** eventIn specifies a list of objects that shall be removed from the **OrderedGroup** node.

The **children** field is the current list of objects contained in the **OrderedGroup** node.

When the **order** field is empty (the default) children are layered in order, first child to last child, with the last child being rendered last. If the **order** field contains values, one value is assigned to each child. Entries in the **order** field array match the child in the corresponding element of the **children** field array. The child with the lowest order value is rendered before all others. The remaining children are rendered in increasing order. The child corresponding to the highest **order** value is rendered last. If there are more children than entries in the **order** field, those children that do not have a drawing order are drawn in the order in which they appear in the **children** field, but after the ones that have an entry in the **order** field.

If there are more order entries than children, the excess order entries are ignored.

Since 2D shapes have no z value, this is the sole determinant of the visual ordering of the shapes. However, when the **OrderedGroup** node is used with 3D shapes, its ordering mechanism shall be used in place of the natural z order of the shapes themselves. The resultant image shall show the shape with the highest **order** value on top, regardless of its z value. However, the resultant z-buffer contains a z value corresponding to the shape closest to the viewer at that pixel. The **order** shall be used to specify which geometry should be drawn first, to avoid conflicts between coplanar or close polygons.

NOTE — Content authors must use this functionality carefully since, depending on the **Viewpoint**, 3D shapes behind a given object in the natural z order may appear in front of this object.

1.2.16 PlaneSensor2D

PlaneSensor2D {

exposedField	SFBool	**autoOffset**	TRUE
exposedField	SFBool	**enabled**	TRUE
exposedField	SFVec2f	**maxPosition**	0, 0
exposedField	SFVec2f	**minPosition**	0, 0
exposedField	SFVec2f	**offset**	0, 0
exposedField	SFNode	**metadata**	NULL
eventOut	SFBool	**isActive**	
eventOut	SFVec2f	**trackPoint_changed**	
eventOut	SFVec2f	**translation_changed**	

}

This sensor detects pointer device dragging and enables the dragging of objects on the 2D rendering plane. The semantics of **PlaneSensor2D** are a restricted case for 2D of the semantics for the **PlaneSensor** node.

1.2.17 PointSet2D

PointSet2D {

exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**metadata**	NULL

}

This is a 2D equivalent of the **PointSet** node with semantics that are the 2D restriction of that node.

1.2.18 Rectangle

Deprecated: Use Rectangle2D instead

Rectangle {

exposedField	SFVec2f	**size**	2, 2
exposedField	SFNode	**metadata**	NULL

}

This node specifies a rectangle centered at (0,0) in the local coordinate system. The **size** field specifies the horizontal and vertical size of the rendered rectangle.

1.2.19 Sound2D

Sound2D {

exposedField	SFFloat	**intensity**	1.0
exposedField	SFVec2f	**location**	0,0
exposedField	SFNode	**metadata**	NULL
exposedField	SFNode	**source**	NULL
field	SFBool	**spatialize**	TRUE

}

The **Sound2D** node relates an audio BIFS sub-graph to the other parts of a 2D audio-visual scene. It shall not be used in 3D contexts (see 9.2.2.1). By using this node, sound may be attached to a group of visual nodes. By using the functionality of the audio BIFS nodes, sounds in an audio scene may be filtered and mixed before being spatially composed into the scene.

The **intensity** field adjusts the loudness of the sound. Its value ranges from 0.0 to 1.0, and this value specifies a factor that is used during the playback of the sound.

The **location** field specifies the location of the sound in the 2D scene.

The **source** field connects the audio source to the **Sound2D** node.

The **spatialize** field specifies whether the sound shall be spatialized on the 2D screen. If this flag is set, the sound shall be spatialized with the maximum sophistication possible. The 2D sound is spatialized assuming a distance of one meter between the user and a 2D scene of size 2m x 1.5m, giving the minimum and maximum azimuth angles of –45° and +45°, and the minimum and maximum elevation angles of -37° and +37 °.

The same rules for multichannel audio spatialization apply to the **Sound2D** node as to the **Sound** (3D) node. Using the **phaseGroup** flag in the **AudioSource** node it is possible to determine whether the channels of the source sound contain important phase relations, and that spatialization at the terminal should not be performed.

As with the visual objects in the scene (and for the **Sound** node), the **Sound2D** node may be included as a child or descendant of any of the grouping or transform nodes. For each of these nodes, the sound semantics are as follows.

Affine transformations presented in the grouping and transform nodes affect the apparent spatialization position of spatialized sound.

If a transform node has multiple **Sound2D** nodes as descendants, then they are combined for presentation as described in **Sound**. If **Sound** and **Sound2D** nodes are both used in a scene, all shall be treated the same way according to these semantics.

1.2.20 Transform2D

Transform2D {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	SFVec2f	**center**	0, 0
exposedField	MFNode	**children**	[]
exposedField	SFNode	**metadata**	NULL
exposedField	SFFloat	**rotationAngle**	0.0
exposedField	SFVec2f	**scale**	1, 1
exposedField	SFFloat	**scaleOrientation**	0.0
exposedField	SFVec2f	**translation**	0, 0

}

The **Transform2D** node allows the translation, rotation and scaling of its 2D children objects.

The **rotation** field specifies a rotation of the child objects, in radians, which occurs about the point specified by **center**.

The **scale** field specifies a 2D scaling of the child objects. The scaling operation takes place following a rotation of the 2D coordinate system that is specified, in radians, by the **scaleOrientation** field. The rotation of the coordinate system is notional and purely for the purpose of applying the scaling and is undone before any further actions are performed. No permanent rotation of the co-ordinate system is implied.

The **translation** field specifies a 2D vector which translates the child objects.

The scaling, rotation and translation are applied in the following order: scale, rotate, translate.

The **children** field contains a list of zero or more children nodes which are grouped by the **Transform2D** node.

The **addChildren** and **removeChildren** eventIns are used to add or remove child nodes from the **children** field of the node. Children are added to the end of the list of children and special note should be taken of the implications of this for implicit drawing orders.

If some of the child subgraphs contain audio content (i.e., the subgraphs contain **Sound** nodes), the child sounds are transformed and mixed as follows.

If each of the child sounds is a spatially presented sound, the **Transform2D** node applies to the local coordinate system of the **Sound2D** nodes to alter the apparent spatial location and direction. If the children are not spatially presented but have equal numbers of channels, the **Transform2D** node has no effect on the childrens' sounds. After any such transformation, the combination of sounds is performed as described in **Sound2D**.

If the children are not spatially presented but have equal numbers of channels, the **Transform** node has no effect on the children's sounds. The child sounds are summed equally to produce the audio output at this node.

If some children are spatially presented and some not, or all children do not have equal numbers of channels, the semantics are not defined.

1.3 Octaga specific nodes

The following nodes are octaga specific and will work in other players.

1.3.1 AvatarNode

AvatarNode {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**metadata**	NULL
field	SFVec3f	**bboxCenter**	**0 0 0**
field	SFVec3f	**bboxSize**	-1 -1 -1
exposedField	SFRotation	**cameraOrientation**	0 1 0 0
exposedField	SFVec3f	**cameraOffset**	0 0 0
exposedField	SFBool	**headPan**	FALSE
exposedField	SFBool	**headTilt**	FALSE
exposedField	SFBool	**headRoll**	FALSE

}

The **AvatarNode** can be inserted in the **avatar** field in a **NavigationInfo** node. It is a groupingNode containing the geometry of an avatar to be used in the scene. The origo of the avatar geometry will be placed in the position of the eye. The **cameraOrientation** and **cameraOffset** allows the camera to be placed relative to the origo of the avatar. The **headPan**, **headTilt**, **headRoll** fields specify wheter the camera or the avatar should move when the user performs the corresponding navigation.

1.3.2 Connection

Connection {

exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
eventOut	SFBool	**isActive**	
exposedField	MFString	**url**	""
exposedField	SFInt32	**protocol**	0
exposedField	SFTime	**timeOut**	0
exposedField	SFBool	**secure**	FALSE
exposedField	SFBool	**noDelay**	FALSE

}

The **Connection** node is used together with one or more **NetworkSensor** nodes to distribute events over the network to other instances of OctagaPlayer connected to the same OctagaCollaborationServer. The **enabled** and **isActive** fields are self-explanatory. The **protocol** and **secure** fields are currently ignored. The **url** should specify the protocol, host, and port

number. (ex: octp://test.octaga.no:3000. Here the protocol is "octp", the server hostname is test.octaga.no and the port is 3000). The **noDelay** disables nagels algorithm on the tcp/ip connection allowing lower latency messages but less optimal bandwith usage.

1.3.3 DropSensor

DropSensor {

exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
eventOut	SFVec3f	**hitPoint**	
eventOut	SFVec3f	**hitNormal**	
eventOut	SFVec2f	**hitTexCoord**	
eventOut	SFTime	**dropTime**	
eventOut	MFURL	**url**	

}

enabled indicates whether the sensor is currently paying attention to pointing device input.
hitPoint the location on the surface of the underlying geometry at which the primary button of the pointing device was released .
hitNormal the normal at the point given by hitPoint .
hitTexCoord the texture coordinate at the point given by hitPoint .
dropTime the time at which the primary button of the pointing device was released.
url returns the URL for the object (resource) currently dragged to the 3D window .

1.3.4 IndexedLineSetAdjacency

IndexedLineSetAdjacency {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
exposedField	MFNode	**attrib**	NULL
exposedField	SFNode	**color**	NULL
exposedField	SFNode	**coord**	NULL
exposedField	SFNode	**fogCoord**	NULL
exposedField	SFNode	**metadata**	NULL
field	MFInt32	**colorIndex**	[]
field	SFBool	**colorPerVertex**	TRUE
field	MFInt32	**coordIndex**	[]

}
The **IndexedLineSetAdjacency** is similar to the IndexedLineSet node except it contains adjacency information for each line segment. That is for each line segment the first and last point is not used for drawing but only for adjacency info. This node is useful for geometry shaders.

1.3.5 JoystickSensor

JoystickSensor {

exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
field	SFInt32	**deviceId**	-1
eventOut	MFBool	**buttonPressed**	
eventOut	SFString	**description**	
eventOut	SFBool	**hasPOV**	
eventOut	SFInt32	**numButtons**	
eventOut	SFInt32	**numAxis**	
eventOut	SFFloat	**pointOfView**	
eventOut	SFFloat	**x**	

eventOut	SFFloat	**y**
eventOut	SFFloat	**z**
eventOut	SFFloat	**r**
eventOut	SFFloat	**u**
eventOut	SFFloat	**v**

}

The **JoystickSensor** node generates events when the user uses a standard joystick or comapatible device (including gamepads). A **JoystickSensor** node can be enabled or disabled by setting the *enabled* field. If the **JoystickSensor** node is disabled, it does not track joystic input or send events. The **deviceId** field can be used to select a specific device connected to the computer using a 0 based index to the. If the deviceId is set to -1 the first available device is used.

The **buttonPressed** field reflects the states of the buttons on the device . The length of the array will be equal to the number of buttons on the device. The value for each button will be TRUE if the button is pressed and FALSE otherwise. The **numButtons** fields also gives the number of buttons on the device.

The **description** field return a textual description of the device.

If the **hasPOV** field is TRUE the device has a means of controlling the point of view which is returned in the **pointOfView** field as a value between 0 and 2*PI. The value can also be -1 which means it is not selected.

The **numAxis** field returns the degrees of movement freedom the user can control using the device. The **x,y,z,r,u,v** fields contains the current value (ranging form -1 to 1) for each of the valid axis. (Example if numAxis is 4 then x,y,z, and r will contain valid values)

1.3.6 MouseSensor

MouseSensor {

exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
eventOut	SFBool	**leftButton**	
eventOut	SFBool	**middleButton**	
eventOut	SFBool	**rightButton**	
eventOut	SFVec2f	**position**	
eventOut	SFVec2f	**windowPosition**	
eventOut	SFFloat	**mouseWheel**	
eventOut	SFBool	**pointingSensorActive**	
eventOut	SFBool	**navigationActive**	

}

The **MouseSensor** node generates events when the user uses a standard mouse or comapatible device. A **MouseSensor** node can be enabled or disabled by setting the *enabled* field. If the **JoystickSensor** node is disabled, it does not track mouse input or send events.

The **leftButton, middleButton** and **rightButton** fields reflects the states of the three mouse buttons . The value for each field will be TRUE if the corresponding button is pressed and FALSE otherwise.

The **position** field returns the cursor position in normalized coordinates (x = [-1,1] y = [-aspect,aspect] while the **windowPosition** field returns the cursor positions in pixels from the top left corner.

The **mouseWheel** field indecates the distence rotated with the wheel.

The **pointingSensorActive** and **navigationActive** reflects whether the user is currently using the mouse to navigate or manipulate any pointing device sensor.

1.3.7 NetworkSensor

NetworkSensor {

exposedField	SFFloat	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
eventOut	SFFloat	**isActive**	
field	SFNode	**connection**	NULL
eventIn	SFString	**httpRequest**	
eventOut	MFString	**httpResponse**	
field	SFString	**networkSensorId**	NULL
# And any number of			
eventIn	fieldType	**set_stateName** / **evt_eventName**	
eventOut	fieldType	**stateName_changed** / **eventName_evt**	

}

The NetworkSensor is used together with the **Connection** node to distribute events over the network to other instances of OctagaPlayer connected to the same OctagaCollaborationServer. The **enabled** and **isActive** fields are self-explanatory. The **connection** field sets the **Connection** node to use. The **networkSensorId** defines a "channel" to commonicate through (It must be matched on the remote client).
The extra user defined fields are used for the actual communication. An event routed to a field called **evt_eventName** distributes the event to all connected clients where the corresponding eventOut named **eventName_evt** is changed. State messages can be distributed by using the in/out pair **set_stateName** / **stateName_changed** with the only difference being that the last transmitted state is stored on the server so that new clients get the correct state upon connecting.

1.3.8 OctagaSensor

MouseSensor {

exposedField	SFNode	**metadata**	NULL
eventOut	SFVec2f	**renderWindowSize**	
eventOut	SFFloat	**nearClip**	
eventOut	SFFloat	**farClip**	

}

The **OctagaSensor** node generates events when certain browser properties is changed. It can be used instead of a script to avoid checking these values every frame. The **renderWindowSize** returns the size of the render window in pixels. The **nearClip** and **farClip** returns the near and far cclip of the scene containing the OctagaSensor node.

1.3.9 PointProperties

PointProperties {

exposedField	SFFloat	**pointSize**	1
exposedField	SFVec3f	**pointAttenuation**	1 0 0
exposedField	SFFloat	**pointFadeThreshold**	1
exposedField	SFNode	**metadata**	NULL

}

The **PointProperties** node allows the point size to be set and to be dependant of the distance from the eye according to the following algorithm:

```
derived_size = size * sqrt(1/(a + b * d + c * d^2))
if (derived_size >= threshold)
  diameter =  derived_size
else
  diameter =  threshold
```

where:
size = **pointSize**
d = the distance from the eye to the point in eye coordinates
a, b , c = **pointAttenuation**
threshold = **pointFadeThreshold**

1.3.10 ReflectionTexture

ReflectionTexture {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**fog**	NULL
exposedField	SFNode	**metadata**	NULL
exposedField	SFInt32	**pixelSize**	-1
exposedField	SFVec3f	**reflectionNormal**	0 1 0
exposedField	SFVec3f	**reflectionPoint**	0 0 0
exposedField	SFString	**update**	“ALWAYS”
exposedField	MFNode	**textureProperties**	[]
exposedField	SFInt32	**samples**	**0**

}

The **ReflectionTexture** node implemts a texture that contains the reflection of a given sub scene as seen from the current point of view reflected on a plane through **reflectionPoint** with normal equal to **reflectionNormal**. The texture created is square whith each side beeing **pixelSize** pixels long. If pixelSize equals -1 (the default) the texture size is dependent on the window size.

The **children** field contain the subscen to be reflected.

The **update** field allows the user to request a regeneration of the texture. Setting this field to "ALWAYS" will cause the texture to be rendered every frame. A value of "NONE" will stop rendering so that no further updates are performed even if the contained scene graph changes. When the value is set to "NEXT_FRAME_ONLY", it is an instruction to render the texture at the end of this frame, and then not render it again. In this case, the update frame indicator is set to this frame; at the start of the next frame, the update value shall be automatically set back to "NONE" to indicate that the rendering has already taken place. Since this is a change of value for the **update** field, an output event is automatically generated.

The **samples** field allows the generated texture to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

The **background** and **fog** fields represent the current values of the bindable children nodes.

textureProperties allows fine control over a texture's application. Note that the for a ReflectionTexture only the following values are valid for the **format** field of the TextureProperties nodes:

Format	Description
“RGBA”	RGBA, 8 bits per component (32 bits per pixel)
“RGBA32F”	RGBA, floating point values, 32 bits per component (128 bits per pixel)
“DEPTH”	DEPTH format

If specifying more than one TextureProperty node Octaga will create multiple output buffers. Theese buffers can be adressed by shaders inside the children field. To get Access to the differnt output buffers use the RenderBuffer node.

1.3.11 RefractionTexture

RefractionTexture {

eventIn	MFNode	**addChildren**	
eventIn	MFNode	**removeChildren**	
exposedField	MFNode	**children**	[]
exposedField	SFNode	**background**	NULL
exposedField	SFNode	**fog**	NULL
exposedField	SFNode	**metadata**	NULL
exposedField	SFInt32	**pixelSize**	-1
exposedField	SFString	**update**	“ALWAYS”
exposedField	MFNode	**textureProperties**	[]
exposedField	SFInt32	**samples**	0
exposedField	SFBool	**depthTexture**	FALSE (deprecated)
exposedField	SFBool	**mapToSingleObject**	TRUE

}

The **ReflectionTexture** node implements a texture that contains a given sub scene as seen from the current point of view. The texture created is square whith each side beeing **pixelSize** pixels long. If pixelSize equals -1 (the default) the texture size is dependent on the window size.

The **children** field contain the subscene to be rendered.

The **update** field allows the user to request a regeneration of the texture. Setting this field to "ALWAYS" will cause the texture to be rendered every frame. A value of "NONE" will stop rendering so that no further updates are performed even if the contained scene graph changes. When the value is set to "NEXT_FRAME_ONLY", it is an instruction to render the texture at the end of this frame, and then not render it again. In this case, the update frame indicator is set to this frame; at the start of the next frame, the update value shall be automatically set back to "NONE" to indicate that the rendering has already taken place. Since this is a change of value for the **update** field, an output event is automatically generated.

The **mapToSingleObject** field is an optimization field that optimizes the the generated texture to be displayed on the single shape that it is applied to. If you want to USE the refraction texture on multiple shapes this flag must be disabled.

The **samples** field allows the generated texture to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

The **background** and **fog** fields represent the current values of the bindable children nodes.

textureProperties allows fine control over a texture's application. Note that the for a RefractionTexture only the following values are valid for the **format** field of the TextureProperties nodes:

Format	Description
“RGBA”	RGBA, 8 bits per component (32 bits per pixel)
“RGBA32F”	RGBA, floating point values, 32 bits per component (128 bits per pixel)
“DEPTH”	DEPTH format

If specifying more than one TextureProperty node Octaga will create multiple output buffers. Theese buffers can be adressed by shaders inside the children field. To get Access to the differnt output buffers use the RenderBuffer node.

The **depthTexture** field is a deprecated way of creating a depth texture. Use textureProperies with format set to DEPTH instead,

1.3.12 RenderBuffer

RenderBuffer {

exposedField	SFInt32	**bufferId**	0
exposedField	SFNode	**metadata**	NULL
exposedField	SFNode	**renderTexture**	NULL

}

The RenderBuffer is used as an output buffer for "render to texture" nodes such as CompositeTexture2D/3D. It can then be used as a normal texture for texturing operations or as input to shaders. The **bufferId** is a zero based index that identifies the RenderBuffer as one of the output buffers from the "render to texture" node specified in the **renderTexture** field. To have more than one output buffer from one "render to texture" this node must specify multiple Textureproperty nodes in its textureProperty field. The "render to texture" node can be USE'd by several RenderBuffers specifying different bufferIds.

1.3.13 Shadow

Shadow {

exposedField	SFNode	**metadata**	NULL	
exposedField	MFNode	**occluders**	[]	
exposedField	SFNode	**light**	NULL	
exposedField	MFNode	**receivers**	[]	
exposedField	SFFloat	**opacity**	0.7	
exposedField	SFInt32	**pixelSize**	-1	
exposedField	SFInt32	**detail**	-1	(deprecated)
exposedField	SFFloat	**scale**	1	
exposedField	SFFloat	**bias**	4	
exposedField	SFInt32	**samples**	**0**	

}

The Shadow node is defined as a grouping node. It generates a shadow based on the occluders and the light source, and applies it to the receivers.

The occluders fields specifies a list of the nodes that shall cast shadow, whereas the receivers fields specifies a list of the nodes that shall receive shadow.

The light field specifies the light source of the shadow. This node must be of type DirectionalLight, PointLight or SpotLight.

The opacity field specifies the transparency of the shadow. The value ranges from 0 to 1. A value of 0 causes an invisible shadow, whereas a value of 1 results in a completely black shadow. The

default value is 0.7.

The shadow created is square texture whith each side beeing **pixelSize** pixels long. If pixelSize equals -1 (the default) the texture size is dependent on the window size.
The detail field is a deprecated way of specifying the resolution of the shadow. Use pixelSize instead as this field will be removed in a future version.

Scale and bias affects the shadow by offsetting the depth values of the occluders. The value of the offset is scale * Δz + r * bias, where Δz is a measurement of the change in depth relative to the screen area of the polygon, and r is the smallest value that is guaranteed to produce a resolvable offset for a given implementation. The offset is added before creating the shadowmap and can be used to avoid numerical errors in situation where the occluder and receiver are very close to or overlapping each other.

The **samples** field allows the shadow to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

1.3.14 ShadowTexture

Shadow {

exposedField	SFNode	**metadata**	NULL
exposedField	MFNode	**occluders**	[]
exposedField	SFNode	**light**	NULL
exposedField	SFInt32	**pixelSize**	-1
exposedField	SFFloat	**scale**	1
exposedField	SFFloat	**bias**	4
exposedField	SFInt32	**samples**	0
exposedField	SFNode	**textureProperties**	[]

}

The ShadowTexture node defines a depth texture for creating shadows in fragmentShaders. It generates a shadow texture based on the occluders and the light source.

The occluders fields specifies a list of the nodes that shall cast shadow.

The light field specifies the light source of the shadow. This node must be of type DirectionalLight, PointLight or SpotLight.

The texture created is square with each side being **pixelSize** pixels long. If pixelSize equals -1 (the default) the texture size is dependent on the window size.

Scale and bias affects the shadow by offsetting the depth values of the occluders. The value of the offset is scale * Δz + r * bias, where Δz is a measurement of the change in depth relative to the screen area of the polygon, and r is the smallest value that is guaranteed to produce a resolvable offset for a given implementation. The offset can be used to avoid numerical errors in situation where the occluder and receiver are very close to or overlapping each other.

textureProperties allows fine control over a texture's application. Note that the for a ShadowTexture only the following values are valid for the **format** field of the TextureProperties nodes:

Format	**Description**
“DEPTH”	DEPTH format

The **samples** field allows the shadow texture to be multisampled with a spcified number of samples per pixel, if samples is 0 no multisampling is performed.

1.3.15 WebSocketSensor

WebSocketSensor {

eventIn	SFString	**send**	
exposedField	SFBool	**enabled**	TRUE
exposedField	SFNode	**metadata**	NULL
field	SFInt32	**port**	3000
eventOut	SFBool	**isActive**	
eventOut	MFString	**message**	

}

The WebSocketSensor works as a server for that accepts webSocket connections. The **enabled** field enables and disables the node. The **port** field defines the port number to listen to. Strings can be sent to the connected clients by routing to the **send** field. The **isActive** field sends TRUE when a client is connected, and FALSE when the last client is disconnected. Messages from the connected clients can be received through the **message** field. If more than one message is received in a single simulation tick, the **message** field will contain multiple strings.