Universidad de las Ciencias Informáticas

Entornos Interactivos 3D (VERTEX)

Facultad 5

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

ENTIDAD BSPLINE PARA LA HERRAMIENTA ASIXMEC.

Autor: Luis Enrique Mederos López del Castillo.

Tutores: Ing. José Ángel Lores Estrada.

Declaración de autoría

Declaró ser el único autor del presente trabajo “Entidad BSpline para la herramienta AsiXMec” y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmó la presente a los ___ días del mes de _____ del año _____.

_____

Firma del Autor

Luis Enrique Mederos López Del Castillo.

_____

Firma del Tutor

Ing. José Ángel Lores Estrada.

Pensamiento

"…aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de sí mismos…"

Ernesto Che Guevara.

Dedicatoria

Dedico este trabajo de diploma a las dos personas más importantes de mi vida: Mis padres, los cuales han dedicado toda su vida al bienestar de sus hijos, siempre tratando de educarnos de la mejor manera para poder ser buenas personas en el futuro, gracias a ellos yo estoy aquí, sin ellos no hubiese sido posible.

Gracias por existir.

Luis Enrique Mederos López Del Castillo.

Resumen

El presente documento aborda la investigación para desarrollar una entidad BSpline que permita modelar el trabajo con curvas abiertas no uniformes del Centro Vertex perteneciente a la Universidad de las Ciencias Informáticas. Para lograr este propósito se analizaron e implementaron en los diferentes módulos de AsiXMec las clases y métodos pertinentes con el fin de que la herramienta fuera capaz de trabajar sobre estas curvas. Además se guió el desarrollo de la herramienta por la metodología AUP, el marco de trabajo de programación Qt y el lenguaje de programación C++.

Índice

Introducción. 1

Capítulo 1: Fundamentación teórica 4

1.1 Introducción. 4

1.2 Introducción a los sistemas CAD. 5

1.3 Función BSpline. 6

1.3.1 Ejemplos de BSplines. 8

1.3.2 Propiedades de las funciones BSpline. 8

1.4 Transformaciones. 9

1.5 Restricciones. 12

1.6 Análisis de Sistemas similares para el tratamiento de curvas BSpline. 14

1.7 Herramientas y tecnologías. 15

1.7.1 Lenguaje UML. 16

1.7.2 Herramienta Visual Paradigm. 16

1.7.3 Lenguaje de programación utilizado C++. 17

1.7.4 Entorno y marco de trabajo *QTCreator*. 18

1.7.5 *OpenCascade*. 19

1.7.6 Framework OCAF. 20

1.7.7 Metodología de desarrollo de *software*. 20

Conclusiones del Capítulo. 22

Capítulo 2 Propuesta de solución 23

2.1 Introducción 23

2.2 ¿Qué es el modelo de dominio? 23

2.3 Soluciones técnicas 25

2.3.1 Requisitos funcionales 25

2.3.2 Requisitos no funcionales 26

2.4 Historia de Usuario para los requisitos funcionales de la aplicación. 27

2.5 Patrones de arquitectura de *software* 31

2.5.1 Patrón utilizado 32

2.5.2 Solución arquitectónica del *software* 34
2.6 Patrones de diseño 35
2.6.1 Patrones utilizados 35
Conclusiones del capítulo 36
Capítulo 3: Características de la solución propuesta 37
3.1 Introducción 37
3.2 Estilo de código 37
3.3 Descripción de los principales módulos del diseño 39
3.3.1 Módulo Entity 39
3.3.2 Módulo Pattern-2d 40
3.3.3 Módulo *Constraint* 41
3.3.4 Módulo *transformation* 42
Conclusiones del capítulo 43
Capítulo 4: Pruebas a la herramienta 44
4.1 Pruebas de Aceptación 44
4.1.1 Sumario de evaluación de las pruebas 50
4.1.2 Cobertura de las pruebas 50
4.1.3 Consideraciones finales 51
Conclusiones 52
Recomendación 53
Referencias 54

Introducción.

Desde sus inicios el hombre buscó formas para representar visualmente la realidad en que vivía. Luego de muchos años de evolución, con la aparición de las artes plásticas, las matemáticas, y las ingenierías afines; se fue perfeccionando la forma de representación de nuestro entorno; de modo que se fueron ampliando los medios para ofrecer cualquier tipo de información visual. Con la aparición de las Tecnologías de la Información y las Comunicaciones (TICs) se alcanzó un nuevo paradigma de representación visual, conocido como Diseño Asistido por Computadoras (CAD), dándole lugar al surgimiento de la informática gráfica (IG), lográndose una forma de diseñar y modelar sólidos en dos o tres dimensiones (2d o 3d).

El diseño y la fabricación asistidos por computadoras han alcanzado un gran nivel de desarrollo e implantación y se han convertido en una necesidad esencial para la supervivencia de las empresas en un mercado cada vez más competitivo. El uso de estas herramientas permite reducir costes, acortar tiempos y aumentar la calidad de los productos fabricados. Estos son los tres factores críticos que determinan el éxito comercial de un producto en la situación social actual en la que la competencia es cada vez mayor y el mercado demanda productos de mayor calidad y menor tiempo de vida. Un ejemplo sencillo y evidente de estas circunstancias es la industria de la automoción, donde cada día aparecen nuevos modelos de coches con diseños cada vez más sofisticados y se reduce la duración de un modelo en el mercado, frente a la situación de hace unas pocas décadas en las que el número de modelos en el mercado era mucho más reducido y su período de comercialización mucho más largo.

En general las aplicaciones de diseño asistido por computadora más conocidas y utilizadas, se mueven alrededor de licencias del *software* privativo; por lo que esto se convierte en un inconveniente para cualquier institución que económicamente no se pueda permitir los gastos que impone el mercado. Sin embargo, el número de tecnologías destinadas a este sector se ha incrementado de igual manera, tanto en el *software* libre como en el privativo.

Con la aparición de tecnologías de representación gráfica como *OpenGL*[1], *VTK*[2] y *OpenCascade*[3], capaces de sustentar productos con un nivel de competitividad alto, comparables a herramientas privativas, se ha podido dar un impulso al desarrollo de aplicaciones de este tipo bajo la filosofía del *software* libre, y de esta manera, se ha logrado alcanzar una mayor equidad entre las aplicaciones privativas y libres referentes a este campo.

Dentro de las funcionalidades principales de los sistemas de diseño asistido por computadoras se encuentra el tratamiento de las curvas, entidad muy importante que permite lograr mejores esbozos de figuras y piezas. Existen herramientas como *Blender* y *3ds Max*, que son consideradas de vanguardia en este campo, las cuales poseen entre sus funcionalidades el tratamiento de curvas BSplines en 2d. Estas herramientas cuentan con la particularidad de tener puntos de control que permiten mover y modificar a voluntad las curvas generadas, funcionalidad muy importante para los diseñadores y mecánicos. Estas curvas es necesario utilizarlas desde diseños sencillos hasta otros más complejos, siendo de gran utilidad para diseñar un importante número de partes y piezas mecánicas.

Una de las directrices fundamentales del proceso de la informatización que se lleva a cabo en nuestra sociedad, es garantizar la soberanía e independencia tecnológica con el objetivo de satisfacer la demanda de los clientes nacionales.

En nuestro país se encuentra en desarrollado una herramienta de diseño y modelado, la cual debe permitir a los diseñadores las opciones de última generación que marcan las principales tendencias en tecnologías CAD, la misma ha sido desarrollada en la Universidad de las Ciencias Informáticas (UCI), específicamente en el Centro de Entornos Interactivos 3D (VERTEX), cuya primera versión ya fue liberada. En la primera versión no se pueden realizar diseños como los de una tubería, ganchos, bordes de carrocerías, vigas con formas curvas, etc, significando

[1] Biblioteca Gráfica para representar y modelar objetos en 3D, contiene una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.
[2] *Framework* de visualización y modelado 3D *Visualization Toolkit*.
[3] ***OpenCascade*** es un núcleo de CAD de nivel profesional, que cuenta con avanzadas capacidades de manipulación de geometría 3D y objetos.

esto un impedimento significativo a la hora de su comercialización. La realización con éxito de la adición del tratamiento de las curvas en la herramienta AsiXMec podrá garantizar una mejor funcionalidad de la misma y elevaría su calidad como producto. Por tanto, se identifica el siguiente **problema científico:** la imposibilidad de modelar mediante una entidad 2D curvas abiertas no uniformes en el *sketcher* de la herramienta AsiXMec, impide la realización de un gran número de diseños específicos de partes y piezas que utilicen como base este tipo de curvas, quedando estos limitados a líneas rectas, puntos, círculos, arcos, rectángulos y polígonos.

Se define como **objeto de estudio:** BSplines en sistemas CAD, y como **campo de acción:** uso del BSpline en *sketchers* 2d para el tratamiento de curvas.

Para darle solución al problema científico que se plantea se tomó como **objetivo general:** Implementar la entidad BSpline para la herramienta AsiXMec, sus restricciones, y las funcionalidades que son comunes a todas las entidades 2d, así como operaciones básicas sobre entidades 2d.

Las tareas investigativas a realizar para asegurar el cumplimiento del objetivo planteado son:

- Revisión bibliográfica para generar el marco teórico conceptual de la investigación que represente las principales definiciones y conceptos relacionados con el tema, y las tendencias en herramientas similares.
- Realización del levantamiento de requisitos funcionales.
- Definición de la arquitectura y la propuesta de solución.
- Implementación de la entidad, las operaciones básicas sobre ella y las restricciones geométricas aplicables.
- Pruebas a la entidad implementada.

Para guiar la investigación científica se utilizarán los siguientes métodos:

Métodos teóricos:

- **Análisis histórico-lógico:** para el estudio de trabajos similares e investigaciones que abordan el tema de la implementación de la entidad BSpline en herramientas a nivel nacional e internacional, que sirvan como punto de partida para el desarrollo de la solución en su conjunto.
- **Método analítico-sintético:** para analizar desde diferentes aristas los conceptos asociados a herramientas homólogas, curvas BSplines, *sketcher* entre otros y así, sintetizar la información recopilada, permitiendo describir las características generales y las relaciones esenciales entre estas.
- **Método de la modelación:** para crear abstracciones con el objetivo de explicar la realidad, se utilizará para la modelación de los diversos diagramas necesarios en cada uno de los flujos de trabajo según la metodología seleccionada.
- **Métodos empíricos:** El método observación científica se empleará con el objetivo de observar el funcionamiento de algunas entidades BSpline que se encuentran en diversas herramientas similares, y a partir de ellas obtener un registro visual de las características comunes en estas, y ver qué puede formar parte de la solución.

Capítulo 1: Fundamentación teórica

1.1 Introducción.

Para comprender la trascendencia y el alcance del presente trabajo, es necesario conocer el significado de los principales conceptos y terminologías asociadas al dominio del problema; además de comprender con claridad, cada uno de los aspectos que componen la fundamentación teórica. Para sostener lo mencionado anteriormente, en este capítulo se refleja el contexto que rodea al objeto de estudio a investigar.

1.2 Introducción a los sistemas CAD.

La mayoría de los programas de diseño asistido por computadora (CAD, por sus siglas en inglés), gestionan una base de datos de entidades geométricas (puntos, líneas, arcos, etc.) con las que se pueden operar a través de pantallas gráficas en la que estas se muestran. La interacción del usuario se realiza mediante comandos de edición o dibujo, desde las líneas de órdenes a la que estos programas están fundamentalmente orientados. Las versiones modernas de los sistemas CAD permiten la interacción mediante una interfaz, conocida como *sketcher*, donde las herramientas CAD realizan sus operaciones, el mismo visualiza el diseño en el boceto 2d (1).

Dentro de los beneficios que brindan los programas CAD se incluyen menores costos de desarrollo de productos, aumento de la productividad, mejora en la calidad del producto y un menor tiempo de lanzamiento al mercado, así como (1):

- Mejor visualización del producto final, los sub-ensambles parciales y los componentes en un sistema CAD agilizan el proceso de diseño.
- Ofrece gran exactitud, de forma que se reducen los errores.
- Brinda una documentación más sencilla y robusta del diseño, incluyendo geometría y dimensiones, lista de materiales, etc.
- Permite una reutilización sencilla de diseños de datos y mejores prácticas.

Los procesos CAD consisten en cuatro etapas (2).

- Modelado geométrico: Se describe como forma matemática o analítica a un objeto físico, el diseñador construye su modelo geométrico emitiendo comandos que crean o perfeccionan líneas, superficies, cuerpos, dimensiones y texto; los cuales originan a una representación exacta y completa en dos o tres dimensiones.

- Análisis y optimización del diseño: Después de haber determinado las propiedades geométricas, se somete a un análisis ingenieril donde se pueden analizar las propiedades físicas del modelo (esfuerzos, deformaciones, deflexiones, vibraciones).
- Revisión y evaluación del diseño: En esta etapa importante se comprueba si existe alguna interferencia entre los diversos componentes, es útil para evitar problemas en el ensamble y el uso de la pieza.
- Documentación y dibujo *(drafting)*: Por último, en esta etapa se realizan planos de detalle y de trabajo. De esta forma se pueden reproducir en dibujos diferentes vistas de la pieza, manejando escalas en los dibujos y efectuando transformaciones para presentar diversas perspectivas de la pieza.

El diseño asistido por computadora muestra el proceso completo de fabricación de un determinado producto con todas y cada una de sus características como tamaño, contorno, etc. Todo esto se graba en la computadora en dibujos bidimensionales o tridimensionales. De esta forma, si el diseñador o mecánico desea mejorarlos, lo puede realizar con posterioridad, o compartirlos con otros para perfeccionar su diseño. La fabricación de productos por medio del diseño asistido por computadora tiene muchas ventajas respecto a la fabricación con operarios humanos. Entre estas están la reducción de coste de mano de obra, o la eliminación de errores humanos (2).

1.3 Función BSpline.

Una curva BSpline es una curva polinómica a tramos, donde los tramos se conectan entre sí con un cierto grado de diferenciabilidad. El término BSpline fue acuñado por Isaac Jacob Schoenberg y es la abreviatura de spline básica. Las BSplines pueden ser evaluadas de una manera numéricamente estable por el algoritmo de De Boor (3).

El algoritmo de De Boor es una generalización del algoritmo de De Casteljau. Este algoritmo, numéricamente estable, proporciona un método para encontrar un punto

C (*u*) de una curva BSpline a partir de los puntos de control y *u* en su dominio (3).

Algoritmo de De Boor (3):

Sean $P_1, \dots, P_n \in R^2$ los puntos de control de la curva BSpline $y\ u\ \in [U_c, U_{c+1}]$.

1. Para i = 0 hasta n.

   $P_i(0) = P_i$.

2. Si u 6 =U_c , entonces s = 0.
3. Si no se encuentra la multiplicidad s de U_c
4. Para *r* = 1 hasta *p* − *s*.

   Para *i* = *c* − *p* + *r* hasta *c* − *s*.

   $$A_{ri} = (U_i + U_p \pm 1U_i - U_i).$$

   $$P_i(r) = (1 - A_{r-1})P_1(-r - 11) + A_{ri}P_{i(r-1)}.$$

5. Regresar $C(u) = P_{c^{(p)}}$.

Definición. Dado un soporte:

$$S = \{U_0, \dots, U_m\} = \{X_0, X_0, \dots, X_0;\ X_1, X_1, \dots, X_1; \dots; X_n, X_n, \dots X_n\}$$

En el cual las X representan:

- X_0 Representa R_0.
- X_1 Representa R_1.
- X_n Representa R_n.

Donde $m = \sum_{i=0}^{n} r_i - 1$, un BSpline $B\ i^p(x)$ de grado p es una función definida en (−∞, ∞), de clase C^{p-ri} en cada punto de ruptura x_i, y que es nula excepto en p + 1 subintervalos consecutivos, en los cuales tiene grado p.

Observaciones

1. Un BSpline restringido a un subintervalo es un polinomio de grado p.

2. El grado p debe ser siempre mayor o igual a $r_1 - 1$.

3. Un BSpline es nulo en a lo más p + 1 subintervalo (depende de la multiplicidad de los nodos).

1.3.1 Ejemplo de BSpline.

1. BSpline de grado 0.

   Dada la partición $\{0, 1, 2, 3, 4, 5\}$; como p=0; las multiplicidades r_i tienen que cumplir $1 \leq p + 1 = 1$, de aquí que $r_i = 1$. El soporte a considerar es $\{0, 1, 2, 3, 4, 5\}$.

   El orden de continuidad es $r_i = -1$. Es decir, las funciones son discontinuas en cada nodo.

- $B_0^0(x) = \{ c, 0 \leq x < 1$
- $B_1^0(x) = \{ c, 1 \leq x < 2$
- $B_2^0(x) = \{ c, 2 \leq x < 3$
- $B_3^0(x) = \{ c, 3 \leq x < 4$
- $B_4^0(x) = \{ c, 4 \leq x < 5$

1.3.2 Propiedades de las funciones BSpline.

En la práctica, las curvas de Béizer [4] rara vez se utilizan, ya que no proporcionan una aproximación suficientemente exacta para las características de un polígono. Por esta razón, en los años 70 se introdujeron los BSplines paramétricos, y se

[4] Las curvas de Béizer son curvas paramétricas que tienen asociado un polígono de control que permite la modificación local de su forma, lo cual las hace una herramienta útil para el modelado geométrico.

utilizan en lugar de los polinomios de Bernstein[5]. Las funciones paramétricas BSplines se emplean ampliamente en los paquetes de gráficos por ordenador ya que gozan de las siguientes propiedades (4):

- Un BSpline pasa por el primer y último punto de control.
- El control de la curva es local. Cada vértice afecta la forma de la curva en un intervalo dado.
- Una BSpline de orden 3 siempre es tangente a la parte media de los lados del polígono.
- Los BSplines permiten crear curvas con "esquinas". Para ello, bastará con crear vértices múltiples.

Una vez implementada la entidad BSpline, debe ser posible que sobre ella se puedan aplicar todas las operaciones y transformaciones que son comunes al resto de las entidades 2d. A continuación, se describe cuáles son las transformaciones y operaciones.

1.4 Transformaciones.

Las transformaciones más típicas sobre el modelo son las de traslación, escalado y rotación. Por lo general, el objeto es centrado en el punto de coordenadas (0,0) del sistema, para realizar su traslado, escalado y rotación.

Traslación: La traslación de un punto P (x, y) consiste en mover el punto según un desplazamiento T (D_x, D_y): D_x respecto al eje x y D_y respecto al eje y, obteniendo un nuevo punto $P^1(X^1, Y^1)$. Además la traslación admite incrementos y disminuciones en movimientos relacionados con el eje seleccionado (5).

[5]Se le llama polinomios *de* Bernstein de grado *n* a los polinomios dados por la expresión:
$N_K^n(x)\binom{n}{k}x^k(1-x)^{n-k}, k = 0, \dots, n,$

Escalado: Los objetos pueden ser escalados (estrechados o estirados) según un factor S_x respecto al eje x y $S_y$respecto al eje y, es decir que admite incrementos y disminuciones en el tamaño del objeto (5).

Rotación: Un punto puede ser rotado respecto al origen en un ángulo α, ya que permite giros a favor y en contra de las manecillas del reloj. Esta transformación se define matemáticamente como (5):

$$y^{|} = xsen_{\alpha} + ycos_{\alpha}$$

$$X^{|} = xcos_{\alpha} - ysen_{\alpha}$$

En el *sketcher* están definidos de igual forma ciertos patrones, así como ciertas modificaciones que se le pueden aplicar a las entidades representadas. A continuación, se describirán las que se le aplicarán las curvas BSplines:

Patrón rectangular.

El patrón rectangular permite replicar en forma rectangular entidades según la cantidad de copias definidas previamente.

En este patrón el usuario puede realizar copias de la entidad seleccionada definiendo dos líneas, una en el eje x y otra en el eje y. Este patrón no es más que una copia de una operación ya realizada en el modelo, con una disposición determinada por el patrón y este es bidimensional.

En la herramienta AsiXMec el usuario puede aplicar este patrón efectuando los siguientes pasos:

1. Seleccionar la operación Rectangular.
2. Seleccionar entidades (activado por defecto).

3. Seleccionar dirección 1: Un eje correspondiente a una entidad línea en el boceto actual. Al seleccionar la primera dirección ya es posible crear un "patrón lineal".
4. Especificar el número de elementos (n_1) que tendrá el patrón en la dirección 1, serán generadas $n_1 - 1$ copias en esta dirección.
5. Especificar la distancia entre los elementos en la dirección 1.
6. Especificar el número de elementos (n_2) que tendrá el patrón en la dirección 1, serán generadas $n_2 - 1$ copias en esta dirección.
7. Especificar la distancia entre los elementos en la dirección 2.
8. Efectuar la operación (botón Aceptar). Son creadas las nuevas entidades.
9. Cerrar el cuadro de diálogo.

Patrón circular.

El patrón circular permite replicar en forma de circunferencia entidades previamente seleccionadas según el ángulo definido por el usuario.

En este patrón el usuario puede realizar copias de la entidad seleccionada entre él y un punto escogido en el *sketcher*, formando entre estos dos una circunferencia con la cantidad de copias definidas por el usuario.

En la herramienta AsiXMec el usuario puede aplicar este patrón efectuando los siguientes pasos:

1. Seleccionar la operación Circular.
2. Seleccionar entidades (activado por defecto).
3. Seleccionar eje. Seleccionar una entidad punto perteneciente al boceto actual.
4. Especificar el número de elementos (n) que tendrá el patrón, serán generadas n - 1 copias.
5. Especificar el ángulo del patrón.
6. Efectuar la operación (botón Aceptar). Son creadas las nuevas entidades.

7. Cerrar el cuadro de diálogo.

Patrón espejo.

El patrón espejo permite realizar la reflexión respecto a un eje, de un grupo de entidades seleccionadas previamente por el usuario.

En este patrón el usuario podrá realizar una copia de la entidad seleccionada trazando una línea delante de la entidad y automáticamente en el *sketcher* se dibujará una réplica de dicha entidad.

En la herramienta AsiXMec el usuario puede aplicar este patrón efectuando los siguientes pasos:

1. Seleccionar la operación Espejo.
2. Seleccionar entidades (activado por defecto).
3. Seleccionar “línea espejo”. Seleccionar una entidad línea perteneciente al boceto actual.
4. Efectuar la operación (botón Aceptar). Son creadas las nuevas entidades.
5. Cerrar el cuadro de diálogo.

Las restricciones geométricas no son más que un conjunto de condiciones que se aplican a un perfil, relativas a su forma geométrica, que establecen las relaciones entre sus elementos. Por tanto, en la herramienta AsiXMec se llevan a cabo una serie de restricciones, para que no se pierda la forma geométrica, garantizando de esta forma que cada operación que se realice no modifique la forma de la figura.

1.5 Restricciones.

Se denominan restricciones geométricas a las herramientas que permiten limitar la posición y/o forma de cualquier entidad geométrica, en la mayoría de los casos con

respecto a otras entidades. A continuación, se describirán las que se le aplicarán a la entidad BSpline en la herramienta AsiXMec.

Fijo:

Esta restricción garantiza que la posición de la entidad en el espacio quede fijada en su lugar, garantizando con esta que cualquier transformación que se realice en el *sketcher* no afecte a dicha entidad.

La restricción de fijo puede ser aplicada a todas las entidades en la herramienta AsiXMec y puede ser establecida específicamente dentro de la herramienta de la siguiente forma:

1. Activar la operación Crear una operación de fijo.
2. Seleccionar una entidad individual.
3. Presionar tecla Esc para cancelar la operación.

Coincidencia:

En esta restricción se busca hacer coincidir dos puntos, un punto a una entidad o una línea a una entidad. Como su nombre indica los puntos seleccionados o las líneas se harán coincidentes sobre la entidad seleccionada.

La restricción de coincidencia en la herramienta AsiXMec se realiza efectuando los siguientes pasos:

Opción 1:

1. Activar la operación Crear una restricción de coincidencia ().
2. Seleccionar las entidades por separado. El orden de selección no importa.
3. Repetir el paso 2 mientras se desee.
4. Presionar tecla Esc para cancelar la operación.

Opción 2:

1. Seleccionar las entidades, a través de la selección múltiple.
2. Activar la operación Crear una restricción de coincidencia ().

1.6 Análisis de Sistemas similares para el tratamiento de curvas BSpline.

En la actualidad existen múltiples sistemas de diseño asistido por computadoras, encontrándose en la vanguardia de estas, herramientas como *Blender* y *3Ds Max.* Una descripción más detallada de cómo estos tratan este proceso se explicará a continuación:

3ds MAX.

En *3ds Max* no se utilizan las herramientas de nurbs[6] para diseños de ingeniera u otros campos parecidos, sino que tan solo se trata de la visualización de un proyecto, pero no de su construcción.

Sin embargo, el campo de las nurbs trae sus desventajas en *3ds Max* pues la realización de superficies nurbs, resultan lentas de renderizar en aquellos campos donde se requiere un renderizado muy rápido en detrimento de la exactitud de las superficies. En el campo de las nurbs, se pueden establecer dos estrategias de trabajo, las cuales se pueden combinar bastante bien. Estas dos estrategias son el dibujo por Vértices de Control y por Puntos de paso (6).

La manera en que 3ds Max trabaja sobre las curvas aporta ideas, maneras y formas de tratar la entidad a la hora de diseñarla, en nuestra caso nos guiamos en la estrategia del dibujo por Vértices de Control, en la forma en que estas operan la curva obteniendo los puntos de control.

***BLENDER*.**

[6] ***B-splines racionales no uniformes*** o ***NURBS*** (acrónimo inglés de *non-uniform rational BSpline*) es un modelo matemático muy utilizado en la computación gráfica para generar y representar curvas y superficies.

Para el tratamiento de las curvas BSplines en *BLENDER* se maneja mucho la suavidad (*steepness*) de curva, la cual es controlada con la longitud de la agarradera[7], de las cuales existen 4 tipos (7).

- ***Free Handle*** *(*Agarradera libre) (en negro): Esto puede usarse de la manera en la que desee. Hotkey: la Tecla H (cambia entre línea y alineada).
- ***Aligned Handle*** (Agarradera alineada) (en morado): Las agarraderas siempre están en línea recta. *Hotkey:* la Tecla H (cambia entre línea y alineada).
- ***Vector Handle*** (Agarradera vector) (en verde): Ambas partes de una agarradera siempre apuntan a la agarradera previa o a la siguiente. *Hotkey:* la Tecla V.
- ***Auto Handle*** (Agarradera automática) (en amarillo): Esta agarradera tiene automáticamente asignada por *Blender* una longitud y dirección para asegurar el resultado más suave. *Hotkey: SHIFT-H.*

El análisis y comprensión sobre el tratamiento de curvas que realizan herramientas homólogas, ayuda a comprender y a obtener una visión con mayor claridad, así como ideas a desarrollar sobre el tratamiento de las curvas, facilitando así la generación de una mejor representación lógica.

Para implementar la entidad BSpline en el *sketcher* 2d de AsiXMec es necesaria la selección de un grupo de herramientas y tecnologías las cuales son descritas a continuación.

1.7 Herramientas y tecnologías.

El desarrollo de *software* no fuera posible sin las herramientas y tecnologías que le dan soporte, pues de no hallarse estas el trabajo sería engorroso, tedioso y muy

[7] Agarradera o *Handle*: tipo particular de punteros "inteligentes". Los *handles* son utilizados cuando un programa hace referencia a bloques de memoria u objetos controlados por otros sistemas

lento para la obtención de los resultados. Las metodologías y herramientas a utilizar para llevar a cabo la propuesta de solución son las siguientes.

1.7.1 Lenguaje UML.

Este lenguaje es decisivo para el desarrollo de un sistema ya que permite la representación del *software* para su posterior programación por lo que brinda un mapa de desarrollo para los programadores y resto del equipo.

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML) es un lenguaje estándar para escribir planos de *software*. UML puede emplearse para visualizar, especificar, construir y documentar los artefactos de un sistema que implica una gran cantidad de *software* (8).

El Lenguaje Unificado de Modelado es el lenguaje que permite la modelación de sistemas de *software* con tecnología orientada a objetos más conocido y utilizado en la actualidad. Es un lenguaje gráfico que cuenta con un grupo de diagramas, los cuales son utilizados para visualizar, especificar, construir y documentar un sistema de *software* en cada una de las etapas por las que tiene que pasar. Indica qué es lo que supuestamente hará el sistema, pero no cómo lo hará. Para la solución propuesta se utilizará UML 2.0 con el objetivo de representar los diagramas de clases para su posterior implementación (8).

1.7.2 Herramienta Visual Paradigm.

Las herramientas *CASE* (*Computer Aided Software Engineering, Ingeniería de Software* Asistida por Ordenador) son las aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas contribuyen de manera directa en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras (9).

Visual Paradigm para UML es una herramienta profesional fácil de utilizar, que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a la rápida construcción de aplicaciones de calidad a un menor coste. Permite dibujar todos los tipos de diagramas de clases, la realización de ingeniería tanto directa como inversa, generar el código desde diagramas y generar la documentación automáticamente en varios formatos como *Web* o Formato de Documento Portable (.PDF) y el control de versiones. Además, la herramienta es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto. Proporciona también abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. La única desventaja es que es una herramienta propietaria, pero se distribuyen copias gratuitas para la educación. Para la solución propuesta se utilizará *CASE Visual Paradigm* 8.0 con el objetivo de crear los diagramas de clases para su posterior implementación (9).

1.7.3 Lenguaje de programación utilizado C++.

C++ es la evolución de C adaptada a la programación orientada a objetos. Se encuentra entre los más usados para desarrollar aplicaciones gráficas en 2D, por ser los que con más velocidad ejecutan el código, en general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del Lenguaje Ensamblador). Además, tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores (10).

Es considerado como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Este lenguaje de programación posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel (10):

1. Posibilidad de redefinir los operadores (sobrecarga de operadores)
2. Identificación de tipos en tiempo de ejecución.
3. Embeber código ensamblador.

También es importante destacar que se escogió este lenguaje ya que fue el seleccionado en la primera versión ya liberada de la herramienta, y en aras de ganar tiempo se decidió continuar bajo la misma línea, ya que sería engorroso realizar un nuevo producto con un lenguaje distinto.

1.7.4 Entorno y marco de trabajo *QTCreator*.

Los Entornos Integrados de Desarrollo, *IDE* (del inglés: *Integrated Development Environment*) constituyen programas compuestos por un conjunto de herramientas para los programadores que facilitan la escritura de programas. Pueden dedicarse en exclusivo a un sólo lenguaje de programación o bien, pueden utilizarse para varios. Los *IDE* facilitan un ambiente de trabajo amigable. Para el desarrollo de la entidad se decidió usar *QtCreator* como *IDE*. *QtCreator* ofrece un ambiente de desarrollo completo para la creación de aplicaciones *Qt*. Es una herramienta ligera y multiplataforma, con un enfoque estricto hacia las necesidades de los desarrolladores de aplicaciones *Qt*. Las principales características son (11):

1. El editor avanzado de C++, para escribir, editar y navegar por el código fuente sin utilizar el ratón (*mouse*).
2. Interfaz gráfica para la depuración, que incrementa la percepción de la estructura de clases de *Qt*.
3. Integración con *QtDesigner* para editar los archivos de interfaz gráfica de usuario.
4. La herramienta Localizador (*Locator*) para la navegación rápida por archivos de proyecto, funciones, clases e informaciones de ayuda.
5. Integración con *QtAssistant*, facilitando el acceso a la ayuda de la *API Qt* (tipos de datos, funciones) de *Qt* a través de una ayuda sensible al contexto.
6. Uso de diferentes sistemas para el control de versiones como *Git*, *Subversion*, CVS and *Perforce*.

7. Construir y ejecutar proyectos *Qt* con la herramienta multiplataforma *qmake*.

Qt es un marco de trabajo (*framework*) escrito en C++ para el desarrollo de aplicaciones de Interfaz Gráfica de Usuario, el cual sigue la filosofía de *software* "escriba una vez, compile donde quiera" (*write once, compile anywhere*).

Además, amplía las posibilidades del lenguaje C++ con una extensa biblioteca de clases, de uso intuitivo y dividida en módulos, en los cuales se puede encontrar desde programación de interfaces gráficas de usuarios hasta programación de redes, procesamiento de textos enriquecidos, acceso a datos almacenados en varios de los gestores de bases de datos más populares. A esto se le suma su extensa base de datos de ejemplos listos para usar. *Qt* está disponible bajo licencia *LGPL*, permitiendo el desarrollo de *Software* Libre, y si se desea, *software* comercial no libre, en ambos casos sin vernos obligados al pago de licencias o derechos (11) (12).

1.7.5 *OpenCascade*.

La tecnología *OpenCascade* es otra de las soluciones muy utilizadas en la visualización en 2D, además de ser otro producto multiplataforma desarrollado bajo la filosofía del *software* libre y que goza de mucha popularidad entre los programadores. El éxito de *OpenCascade* lo evidencia las propias ventajas que posee. Primeramente, su paquete de clases brinda una serie de servicios tales como: tipos primitivos, cadenas y varios tipos de cantidades. Además, brinda una gestión automatizada de la memoria acumulada, manejo de excepciones, clases de manipulación de agregados de datos y herramientas matemáticas. A la par, esta tecnología proporciona las estructuras de datos para representar modelos geométricos en dos y tres dimensiones, estas funcionalidades se encuentran localizadas en las clases: geometría 2D, geometría 3D, utilidades geométricas y topologías (9).

1.7.6 Framework OCAF.

OCAF es un marco de aplicaciones que aumenta la productividad de desarrollo de manera significativa, así como ofrece una solución para la estructuración y manejo de datos de aplicación, basado en la arquitectura de aplicaciones / documentos. OCAF maneja datos y algoritmos de *OpenCascade*. Además de ser una plataforma que permite acceder al desarrollo rápido de aplicaciones sofisticadas de diseño (9). Dentro de este se puede encontrar servicios como:

1. Gestión documental: una aplicación OCAF gestiona la creación de documentos.
2. Asociatividad entre los datos, en el sentido de compartir datos entre objetos.
3. Deshacer / Rehacer mecanismo aplicado sobre los comandos.
4. Persistencia: guardar y restaurar documentos.
5. Modificación de documento y recálculo de los datos.

1.7.7 Metodología de desarrollo de *software*.

El Proceso Unificado Ágil de *Scott Ambler o Agile Unified Process* (AUP por sus siglas en inglés) es una versión abreviada del Proceso Unificado de Racional (RUP). Este representa de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

Al igual que otras metodologías de desarrollo esta muestra fases de desarrollo que en un principio eran inicio, construcción, elaboración, transición. La universidad hizo adaptaciones para un mejor acoplamiento en la producción quedando la siguiente estructura (13).

Fases de AUP para la UCI (13)**:**

- **Inicio:** Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental

acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

- **Ejecución:** En esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elabora la arquitectura, el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del *software*.

- **Cierre:** En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Conclusiones del Capítulo.

Se efectuó un análisis de los principales conceptos relacionados con el campo del diseño, ingeniería y fabricación asistidos por computadora como vía de solución a las principales problemáticas de la industria moderna. Así como las disímiles formas de llevarlos a la práctica, la biblioteca *OpenCascade* y su *framework* de desarrollo OCAF, de los cuales se conocieron las principales características a tener en cuenta para una mayor comprensión de la solución propuesta. Se sentaron las bases necesarias para llegar a la total comprensión del objeto de estudio y su campo de acción, se mostraron las principales transformaciones y restricciones geométricas para el desarrollo eficaz del producto.

Capítulo 2 Propuesta de solución

2.1 Introducción

En el presente capítulo se adquiere un enfoque práctico del sistema desarrollado. En el mismo se expondrán las reglas del negocio, así como los requisitos funcionales y no funcionales que regirán el desarrollo de la solución al problema, definiendo qué esperan los usuarios de la misma. Partiendo de esto se determinaron las historias de usuario. Además se describieron los procesos de las principales funcionalidades de la entidad.

AsiXMec no cuenta con una entidad que sea capaz de trabajar sobre las curvas b-splines de forma directa en su *sketcher*, considerándose estas importantes a la hora de desarrollar algún modelado, el cual consta de muchos pasos que demandan conocimientos en el área de diseño, creatividad y exactitud, en los cuáles se pierde tiempo. Para resolver este problema se propone desarrollar una entidad que permita al usuario trabajar sobre estas curvas en el *sketcher* de la herramienta.

2.2 Modelo de dominio

El Modelo de Dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema, conceptos del mundo real en lugar de componentes de *software* (14).

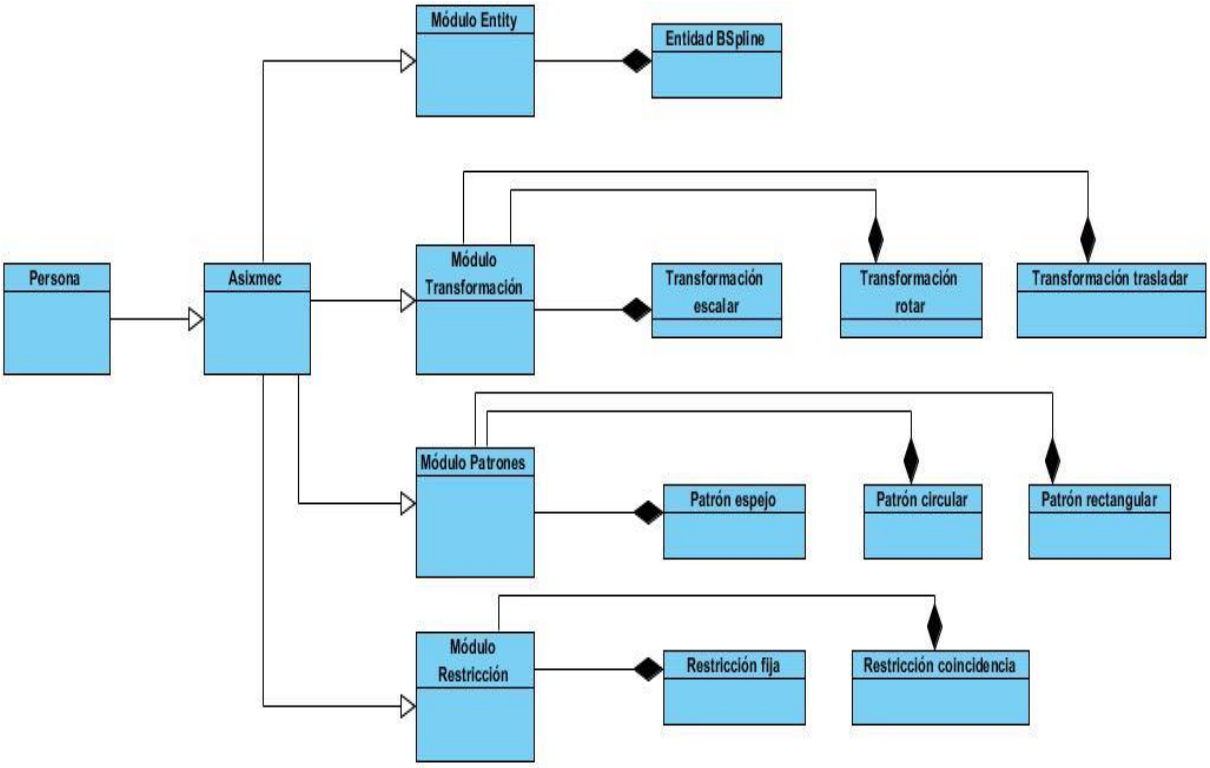

Figura 3: Diagrama del modelo de dominio.

El Diseñador es en este caso el usuario que desea emplear la herramienta AsiXMec para realizar algún diseño empleando la entidad BSpline. Se dirige al módulo *Entity*, Donde debe seleccionar de entre las distintas opciones disponibles la de crear una entidad BSpline. A continuación debe ir seleccionando mediante el *clic* en el *sketcher* los diferentes puntos de control y se va previsualizando la entidad. Terminando de crear a entidad luego de dar 2 *clics* seguidos en el punto que escoge como punto final de la entidad. El usuario tiene la alternativa, una vez creada la entidad, de aplicarle transformaciones a la misma, operación que se realiza escogiendo la transformación que desee aplicar, hallándose entre estas: el escalado, rotación y traslación. También se encuentra la opción patrón, la cual contiene dentro de sí alternativas como patrón rectangular, circular y espejo, brindándole al usuario la posibilidad de aplicar aquel que sea de su interés. El usuario deberá tener en cuenta además, a la hora de realizar su diseño, las restricciones geométricas aplicables a las curvas BSplines.

Actores:

- **Diseñador:** Usuario que emplea la herramienta AsiXMec.
- **AsiXMec**: Herramienta cad. Para el modelado de piezas mecánicas.
- **Módulo *Entity***: Contiene opciones relacionadas con la creación de entidades, las cuáles se encuentran en el grupo figuras.
- **Módulo Patrones**: Permite realizar copias de entidades en dependencia de un patrón específico seleccionado por el usuario.
- **Módulo restricciones**: Las restricciones geométricas se materializan en objetos de acotación y cuentan con un texto que representa su valor.
- **Módulo transformación**: Permite realizar las transformaciones básicas a las entidades seleccionadas.

2.3 Soluciones técnicas

Para darle cumplimiento al objetivo de este trabajo, se pretende implementar en la herramienta AsiXMec una nueva entidad a la cual deben podérsele aplicar todas las transformaciones, patrones y operaciones comunes a todas las entidades básicas 2d, así como las restricciones geométricas que correspondan a una curva BSpline.

2.3.1 Requisitos funcionales

RF1: Crear Entidad BSpline.
RF2: Aplicar Patrón rectangular.
RF3: Aplicar Patrón circular.
RF4: Aplicar Patrón espejo.
RF5: Aplicar transformación trasladar.
RF6: Aplicar transformación rotar.
RF7: Aplicar transformación escalar.
RF8: Aplicar restricción fijo.

RF9: Aplicar restricción coincidencia.

2.3.2 Requisitos no funcionales

Requerimientos de Restricciones en el diseño e implementación

- El sistema debe ser desarrollado en el lenguaje de programación C++.
- La implementación debe ser desarrollada utilizando el marco de trabajo Qt.

Requerimientos de ayuda y documentación

- Cada una de las etapas del ciclo de vida del proyecto deberá contar con una documentación según la metodología establecida.
- Requerimientos de licencias y patentes.
    - ✓ Se deben utilizar herramientas libres.

Requerimientos de Portabilidad

- El sistema debe funcionar en sistemas de la familia *GNU/Linux.* Específicamente *Ubuntu* en su versión 12.04 o superior.

Requerimientos de Seguridad

- Cuando se intente realizar una transformación no válida, el sistema no permitirá que esta se ejecute.

Requerimientos de *Software*

Sistema operativo *GNU/Linux*, distribución *Ubuntu* y derivados.

2.4 Historias de Usuario

Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos. Cada historia de usuario debe ser limitada, ésta debería poderse escribir sobre una nota adhesiva pequeña (15).

2.4.1 Historia de Usuario para los requisitos funcionales de la aplicación.

Numero: 1.	**Nombre del requisito:** Crear Entidad BSpline.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 30 horas.
Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 40 horas.
Descripción: EL usuario selecciona el ícono BSpline y luego en el *sketcher* se van seleccionando los diferentes puntos de control y se va previsualizando la entidad. En el último punto se deben dar 2 *clics*.	

Tabla 1: Historia de Usuario Crear Entidad BSpline.

Numero: 2.	**Nombre del requisito:** Aplicar Patrón rectangular.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta**.**	**Tiempo Estimado:** 25 horas.

Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 15 horas.
Descripción: El patrón rectangular permite replicar en forma rectangular entidades según la cantidad de copias definidas previamente.	

Tabla 2: Historia de Usuario Aplicar Patrón rectangular.

Numero: 3.	**Nombre del requisito:** Aplicar Patrón circular.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta**.**	**Tiempo Estimado:** 15 horas.
Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 20 horas.
Descripción: El patrón circular permite replicar en forma de circunferencia entidades previamente seleccionadas según el ángulo definido por el usuario.	

Tabla 3: Historia de Usuario Aplicar Patrón circular.

Numero: 4.	**Nombre del requisito:** Aplicar Patrón espejo.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta**.**	**Tiempo Estimado:** 15 horas.

Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 15 horas.
Descripción: El patrón espejo permite realizar la reflexión respecto a un eje, de un grupo de entidades seleccionadas previamente por el usuario.	

Tabla 4: Historia de Usuario Aplicar Patrón espejo.

Numero: 5.	**Nombre del requisito:** Aplicar transformación trasladar.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 20 horas.
Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 10 horas.
Descripción: Esta operación se encarga de como dice su nombre. De trasladar la entidad hacia otro lugar en el *sketcher*.	

Tabla 5: Historia de Usuario Aplicar transformación trasladar.

Numero: 6.	**Nombre del requisito:** Aplicar transformación rotar.
Programador: Luis E. Mederos	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 20 horas.

Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 15 horas.
Descripción: En esta operación se rota la entidad, en el *sketcher*, definiendo un ángulo de rotación dado un punto seleccionado. Previamente como el centro de la rotación.	

Tabla 6: Historia de Usuario Aplicar transformación rotar.

Numero: 7.	**Nombre del requisito:** Aplicar transformación escalar.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 20 horas.
Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 15 horas.
Descripción: Esta operación se encarga de hacer la figura más grande o más pequeña según desee el usuario.	

Tabla 7: Historia de Usuario Aplicar transformación escalar.

Numero: 8.	**Nombre del requisito:** Restricción fijo.
Programador: Luis E. Mederos.	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 45 horas.

Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 30 horas.
Descripción: La restricción de fijo se aplica para que las entidades no puedan ser modificadas.	

Tabla 8: Historia de Usuario restricción fijo.

Numero: 9.	**Nombre del requisito:** Restricción de coincidencia.
Programador: Luis E. Mederos	**Iteración Asignada:** 1.
Prioridad: Alta.	**Tiempo Estimado:** 45 horas.
Riesgo en Desarrollo: Definido por el proyecto.	**Tiempo Real:** 45 horas.
Descripción: La restricción de coincidencia se puede aplicar entre puntos, entre un punto y una línea o entre un punto y un círculo. A continuación se muestran los pasos a seguir para aplicar esta restricción. Esta restricción se encarga de hacer coincidir dos figuras.	

Tabla 9: Historia de Usuario restricción coincidencia.

2.5 Patrones de arquitectura de *software*

1. De acuerdo al *Software Engineering Institute* (SEI), la Arquitectura de *Software* se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos (16).

2. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de *software*, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor (16).

Dentro de los objetivos de los patrones se encuentran (16):

- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

2.5.1 Patrón utilizado

Arquitectura de Capas

La metodología RPM [8] presentada por C. Larman presupone una estructura de tres capas que es típica de los Sistemas de Información. Estas tres capas son (14) (15):

- La capa de la Presentación: Esta capa reúne todos los aspectos del *software* que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios humanos. Estos aspectos típicamente incluyen el manejo y aspecto de las ventanas, el formato de los reportes, menús, gráficos y elementos multimedia en general.
- La capa del Dominio de la Aplicación: Esta capa reúne todos los aspectos del *software* que tienen que automatizar o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

[8] RPM (***R**ecommended **P**rocess and **M**odels*) surgió como guía pedagógica para el desarrollo incremental e interactivo de Sistemas de Información, sin embargo es aplicable al desarrollo de sistemas reactivos de arquitectura de tres capas (presentación, dominio de aplicación, repositorio).

- La capa del Repositorio: Esta capa reúne todos los aspectos del *software* que tienen que ver con el manejo de los datos persistentes, por lo que también se le denomina la capa de las Bases de Datos.

Existen tres propuestas de arquitecturas de capas para Sistemas de Información, donde las capas a veces reciben el nombre de niveles:

- Arquitectura de dos capas.
- Arquitectura de tres capas.
- Arquitectura de cuatro capas.

Todo patrón tiene ventajas y desventajas, en el caso de la arquitectura de capas se encuentran (15):

- Ventajas
    - Reutilización de capas.
    - Facilita la estandarización.
    - Dependencias se limitan a intra-capa.
    - Contención de cambios a una o pocas capas.
- Desventajas
    - A veces no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.
    - Pérdida de eficiencia.
    - Trabajo innecesario por parte de capas más internas o redundante entre varias capas.
    - Dificultad de diseñar correctamente la granularidad de las capas.

2.5.2 Solución arquitectónica del *software*

Teniendo en cuenta las características de la herramienta la arquitectura más adecuada para la representación, es la arquitectura 3 capas, del estilo arquitectónico llamada y retorno[9]:

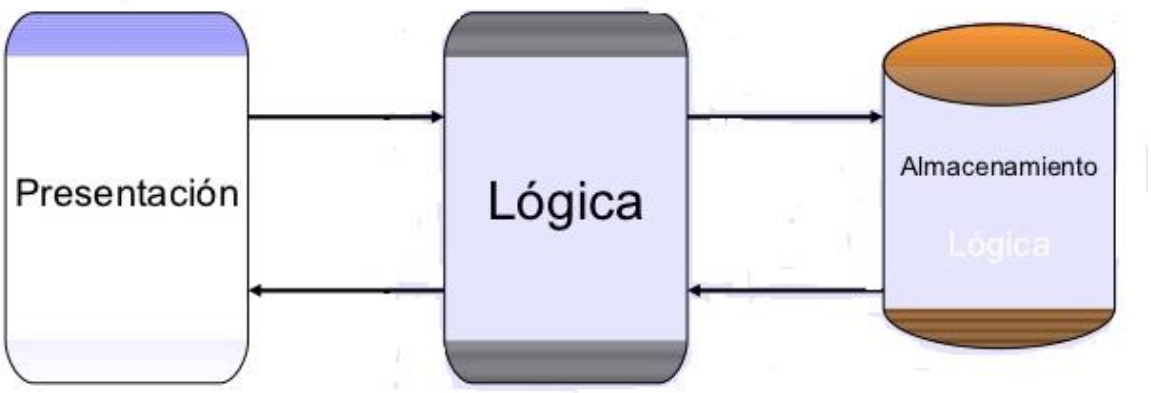

Figura 4: Arquitectura de 3 capas.

La capa de presentación: Es donde se encuentra el *ribbon* para crear las entidades. Y además los iconos de todas las operaciones y restricciones.

La capa lógica: Se encuentran las clases *entity, transformation-plugin, constrain, sweep-plugin, snap, solver-2d-plugin, modify y pattern 2d*.

La capa de almacenamiento: Es donde se almacena el árbol de las operaciones realizadas en la herramienta (los datos persistentes), lo que permite la reutilización del código si es necesario.

[9] Descomposición jerárquica en subrutinas (componentes) que solucionan una tarea o función definida. Los datos son pasados como parámetros y el manejador principal proporciona un ciclo de control sobre las subrutinas. Reflejan la estructura del lenguaje de programación. Permite al diseñador del *software* construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se basan en la bien conocida abstracción de procedimientos/funciones/métodos.

2.6 Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular.

Los patrones de diseño pueden ser:

- **Creacional:** Se encargan de la creación de instancias de los objetos. Abstraen la forma en que se crean los objetos, permitiendo tratar las clases a crear de forma genérica, dejando para después la decisión de qué clase crear o cómo crearla. Según donde se tome dicha decisión se pueden clasificar los patrones de creación en: patrones de creación de clases (la decisión se toma en los constructores de las clases) (17).
- **Estructural:** Son los que plantean las relaciones entre clases, las combinan y forman estructuras mayores. Tratan de conseguir que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y éstas están determinadas por las interfaces que soportan los objetos. Estudian cómo se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos (17).
- **Comportamiento:** Plantea la interacción y cooperación entre las clases. Los patrones de comportamiento estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal. Los patrones de comportamiento son (17).

Durante el desarrollo de la solución fueron utilizados los patrones de tipo Comportamiento, los cuales serán descritos posteriormente.

2.6.1 Patrones utilizados

Experto: Está evidenciado en la forma en que se le asigna a cada una de las clases que conforman la arquitectura de AsiXMec las responsabilidades correspondientes

de acuerdo a los atributos que las mismas manejan, lo que garantiza que se realicen de manera correcta cada una de las operaciones que se aplican sobre la entidad BSpline. Este patrón se evidencia específicamente en las clases *Command* y Controladoras. Las primeras se encargan de guardar los datos como nodos en el árbol de OCAF, mientras que las segundas, de las operaciones que se van a realizar así como de los eventos del *mouse*.

Alta cohesión: Se evidencia en las formas en que diferentes clases que conforman los módulos de AsiXMec cooperan de manera armónica para realizar las operaciones requeridas sobre la entidad BSpline. Ejemplo de esto se ve reflejado en como las clases Controladoras llaman internamente a las clases *Command* para que estás guarden los datos en el árbol de OCAF y luego una vez guardados, las clases *Driver* se encarguen de extraer estos datos para reflejarlos en el *sketcher*, el cual es con el que interactúa el usuario.

Conclusiones del capítulo

En este capítulo se formalizó un análisis critíco a todos los requisitos planteados al Análisis y Diseño propuesto, así como se definieron las principales reglas del negocio a seguir, quedando de esta manera conformado la estructura del negocio.

Capítulo 3: Características de la solución propuesta

3.1 Introducción

En este capítulo se explica la solución desarrollada al problema planteado; brindando una descripción de los diferentes módulos de la aplicación en los que intervienen las curvas BSpline; Así como una descripción de funcionalidades que intervienen en la lógica del negocio, y el estilo de código empleado.

3.2 Estilo de código

Líneas: Se emplea solo una instrucción por línea de código para una mejor legibilidad, comprensión y limpieza en el trabajo.

```
void MirrorPatternController::selectEntities()
{
        selectedNodes.clear();
        selectedEntities.clear();
        mirroredShapes.clear();
```

Figura 5: Ejemplo de estilo de código en cuanto a líneas.

Bloques: Se utilizan llaves para delimitar todos los bloques de sentencias de control y bucles. Las llaves están en la misma columna.

```
for(int i = 0; i < (int) mirroredShapes.size(); i++)
{
    DocumentNodeType type = createdEntities.at(i).type();

    myMap[type]->execute(createdEntities.at(i), mirroredShapes.at(i));

   if(createdEntities.at(i).type() == ARC_NODE_TYPE)
    {
        ArcNode arcNode = (ArcNode&) createdEntities.at(i);

        PointNode firstPointNode = arcNode.firstPointNode();
        PointNode lastPointNode = arcNode.lastPointNode();
        PointNode centerPointNode = arcNode.centerPointNode();

        ModifyArcCommand modifyArcCmd = ModifyArcCommand(arcNode, centerPointNode,
                                                         lastPointNode, firstPointNo
        modifyArcCmd.execute();
    }
}
```

Figura 6: Ejemplo de estilo de código en cuanto a bloques.

Espacios en blanco: Se hace uso generoso de líneas y espacios en blanco en pro de hacer más claro y comprensible el código. Las funciones y los bloques de código son aislados unos de otros, usando varias líneas en blanco para facilitar su lectura y poner de manifiesto su carácter de unidad de programación.

Así se conseguirá tener un código de fácil lectura (si fuera excesivamente compacto prácticamente ilegible), y en el que sea sencillo localizar funciones y bloques de código).

```
void MirrorPatternCommand::execute()
{
    CopyEntities2DCommand copyEntitiesCmd(selectedEntities);
    copyEntitiesCmd.execute();

    createdEntities = copyEntitiesCmd.getCopiedEntities();

    for(int i = 0; i < (int) mirroredShapes.size(); i++)
    {
        DocumentNodeType type = createdEntities.at(i).type();

        myMap[type]->execute(createdEntities.at(i), mirroredShapes.at(i));
```

Figura 7: Ejemplo de estilo de código en cuanto a espacios en blanco.

Comentarios: Los comentarios (pueden ser minimizados), facilitan la comprensión del código y aportan información útil sobre las sentencias, bloques, variables, funciones que afectan.

```
/**
 * @brief Add a point on bspline constraint to solver.
 * @details
 * @retrun
 * @param pointNode and BSplineNode are the entities to which the restriction will apply.
 * @param checkConflict is to know if conflict will be check.
 */
bool addPointOnBSplineConstraint(PointNode& pointNode, BSplineNode& bsplineNode,
                                 bool checkConflict = true);
```

Figura 8: Ejemplo de estilo de código en cuanto a comentarios.

3.3 Descripción de los principales módulos del diseño

3.3.1 Módulo *Entity*

AsiXMec contiene en la ribbon la pestaña Boceto donde se encuentran funcionalidades agrupadas por categorías. Las opciones relacionadas con la creación de entidades, se encuentran en el grupo Figuras. En este caso se explica la creación del BSpline.

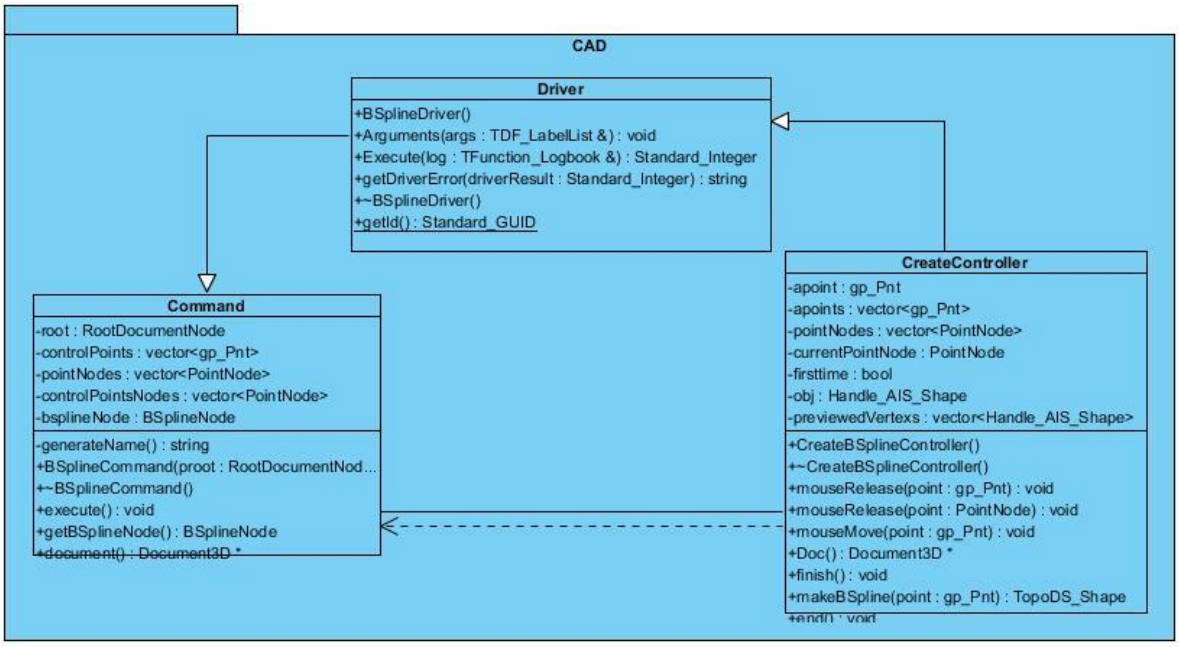

Figura 9: Diagrama de clases del BSpline en el módulo *Entity.*

En este módulo es donde se crea la curva BSpline para luego realizar las operaciones sobre dicha curva para el modelado que se desee realizar. En la clase *CreateBSplineController* es donde se encuentra y recopila toda la información necesaria para crear una entidad BSpline, además de llamar internamente a la clase *BSplineCommand* la cual se encarga de guardar los datos en el árbol de OCAF, para finalmente la clase *BSplineDriver* extraer los datos de dicho árbol, y ejecutar la creación de la entidad que es visible al usuario. En este módulo fue también necesario implementar la clase *ModifyBSplineCommand,* que se encarga de actualizar

en el árbol de OCAF la información de cualquier BSpline cuando este sea modificado. Es importante destacar que la clase driver hereda de la clase *ObjectDriver* que se encuentra dentro del módulo *cad-core* y dicha clase hereda a su vez también de la clase driver de dicho módulo. Así mismo la clase *BSplineCommand* hereda de la clase *Command* que se encuentra dentro del módulo *cad-core*, y la clase *CreateBSplineController* hereda de la clase *CreateEntityController,* la cual hereda de la clase *QObject*.

3.3.2 Módulo Pattern-2d

La pestaña Boceto también contiene la categoría de Patrones, la cual permite realizar copias de entidades en dependencia de un patrón específico seleccionado por el usuario.

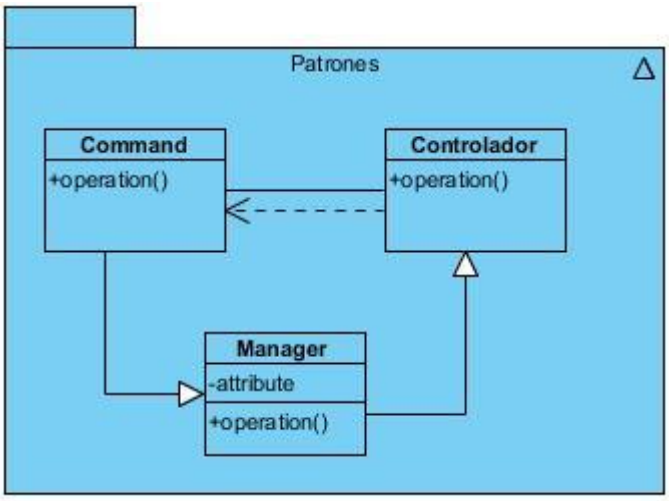

Figura 10: Diagrama de clases del BSpline en el módulo *Pattern-2d.*

En este módulo dependiendo del patrón en específico a utilizar (circular, rectangular y espejo), se ejecuta el comando y la lógica correspondiente, es importante destacar que cada patrón presenta un controlador distinto, esto se debe a que la lógica que se plantea en cada uno es distinta. Este módulo trabaja internamente con los comandos que se encuentran implementados en el módulo *transformation* para eje-

cutar rotaciones (patrón circular), y traslaciones (patrón rectangular). Solo fue necesario realizar modificaciones en el caso del patrón espejo donde fue necesario insertar dentro del mapa de dicho patrón el par <*DocumentNodeType, CommandExecutor*> para en el caso de que la entidad a aplicársele el patrón fuera un BSpline, se llamara adecuadamente al comando que ejecutara el nuevo BSpline obtenido luego de aplicada la operación. Todas las clases *Command* de este módulo heredan de la clase *Command* del módulo *cad-core.*

3.3.3 Módulo *Constraint*

Las restricciones geométricas se materializan en objetos de acotación y cuentan con un texto que representa su valor.

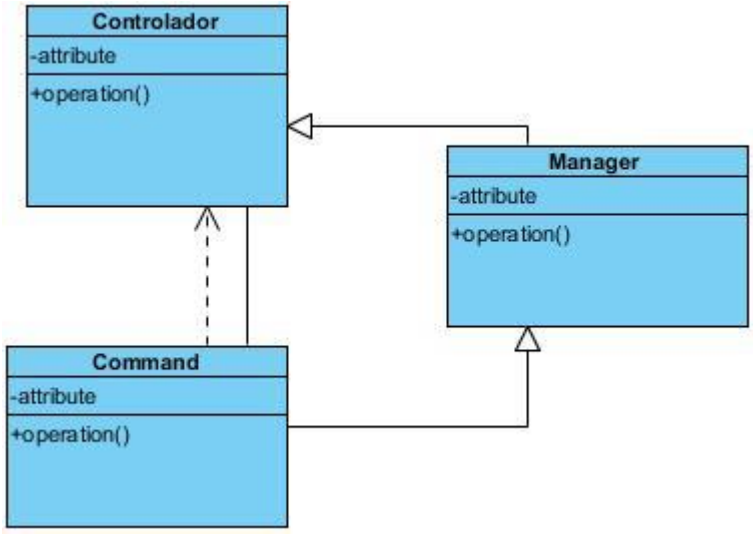

Figura 11: Diagrama de clases del BSpline en el módulo *Constraint.*

En el caso específico de agregar una restricción a una curva BSpline en este módulo, se crea un método (clase *Constraint*), el cual internamente llama al *solver-2d* , dentro del que se realizarán las operaciones necesarias para agregar una clase de tipo *GCS para el BSpline* (*GCS* son las clases que intervienen en las restricciones), para finalmente llamar a la clase *SolverSystem* a través de otro método (*GCSSys*: es la clase que se encarga de los cálculos matemáticos para el cálculo y recálculo de las entidades y restricciones(lo llama la clase *solver*)), aunque por no contar

dicho solver con la implementación de los cálculos necesarios para el BSpline, fue necesario implementar las restricciones a los puntos de control del BSpline, y no al BSpline en sí, para poder lograr el resultado visual del recálculo de la entidad en el *sketcher* al aplicársele una restricción. El proceso descrito anteriormente logra un flujo de datos y operaciones que restringirán a dicha curva.

3.3.4 Módulo *transformation*

En este módulo se altera la figura seleccionada logrando de esta forma que la misma cambie su posición en el *sketcher*, permitiendo representar la misma figura en otra posición aunque es posible dejar la figura estática y crear una copia con las dimensiones escogidas.

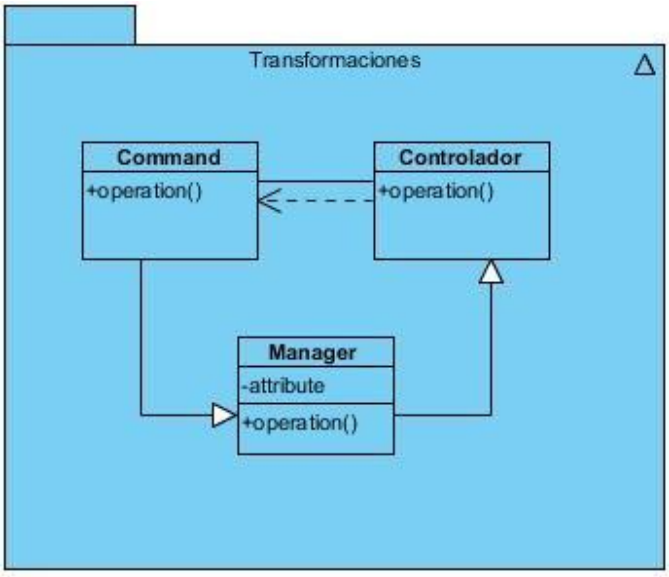

Figura 12: Diagrama de clases del BSpline en el módulo *Transformation.*

Este módulo también tiene como característica que cuenta con clases *Command*, M*anager* y *Controller,* que heredan en cada caso de una clase principal dentro del módulo, y luego se hacen específicas de acuerdo a la transformación a aplicar. La clase *transformationCommand* hereda de la clases *Command* del módulo *cad-core*. En este módulo, en la clase *transformationCommand* se adicionó al mapa un nuevo

elemento cuyo par está conformado por, un miembro del *enum transformationCommand DocumentNodeType* (*BSPLINE_NODE_TYPE*, que también fue necesario añadir), y una llamada al constructor de un comando que se implementó dentro del módulo (*CommandBSplineExecutor*), que se encarga de actualizar en el árbol de OCAF la información del BSpline transformado. En el método *execute* () de dicho comando se utilizó el comando *ModifyBSplineCommand* implementado en el módulo *Entity*.

Conclusiones del capítulo.

Se realizó un análisis de los principales módulos que intervienen en la incorporación de la entidad BSpline, tanto una descripción en la secuencia de flujo, así como las principales clases y métodos que fueron participe en la creación, y tratamiento de la entidad. Permitiendo de esta forma una mayor interpretación sobre la misma.

Capítulo 4: Pruebas a la herramienta

Los casos de prueba son pruebas unitarias o funcionales que se efectúan al sistema. Las pruebas funcionales se realizan por un agente externo a la aplicación, este puede ser un cliente o el usuario al que está dirigido, y las pruebas unitarias son validaciones desde el punto de vista del desarrollador. El objetivo principal de las pruebas es legitimar las historias de usuario. Por lo que los casos de prueba deben acompañar al sistema durante su ciclo de explotación (18).

Usualmente se genera una prueba por cada historia de usuario, aunque no existe un límite definido de la cantidad de pruebas que un sistema debe tener.

4.1 Pruebas de Aceptación

Para las pruebas de la herramienta AsiXMec se decidió aplicar las pruebas de aceptación bajo la siguiente estructura:

Caso de prueba de Aceptación	
Número: N	**Historia de Usuario:** N
Nombre: Nombre	
Condiciones de ejecución: Condiciones	
Entradas/Pasos de ejecución: Pasos	
Resultado esperado: Resultado esperado.	
Evaluación de la prueba: Evaluación.	

Tabla 10: Ejemplo de casos de prueba.

Número: Representa el número del caso de prueba. Este debe ser consecutivo.

Historia de usuario: Número de la historia de usuario a la que responde el caso de prueba.

Condiciones de ejecución: Condiciones previas que deben cumplirse para la realización del caso de prueba.

Entradas/Pasos de ejecución: Secuencia de pasos o entradas que se realizan para validar la funcionalidad que se prueba.

Resultados esperados: Describe los objetivos concretos de la funcionalidad.

Evaluación de la prueba: Evaluación obtenida luego de realizada la prueba. Puede ser satisfactoria o no satisfactoria.

A continuación se muestran algunas de las pruebas de aceptación realizadas a las funcionalidades de la aplicación.

Caso de prueba de Aceptación	
Número: 1.	**Historia de Usuario**: 1.
Nombre: Crear Entidad BSpline.	
Condiciones de ejecución: Sin condiciones.	
Entradas/Pasos de ejecución: 1. El usuario selecciona el icono BSpline que se encuentra en el menú de figuras.	
Resultado esperado: Construcción del BSpline.	
Evaluación de la prueba: Satisfactoria.	

Tabla 11: Caso de prueba a la historia de usuario 1.

Caso de prueba de Aceptación	
Número: 2.	**Historia de Usuario**: 2.

Nombre: Aplicar Patrón rectangular.
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción rectangular que se encuentra en el boceto en la parte de patrones. 2. Se traza dos líneas una en el eje x y la otra en la y. 3. Selecciona el eje donde se desee seleccionar las copias.
Resultado esperado: Se aplica el patrón.
Evaluación de la prueba: Satisfactoria.

Tabla 12: Caso de prueba a la historia de usuario 2.

Caso de prueba de Aceptación	
Número: 3.	**Historia de Usuario**: 3.
Nombre: Aplicar Patrón circular.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción circular que se encuentra en el boceto en la parte de patrones. 2. Se selecciona un punto en el *sketcher* para realizar la copia de la entidad.	
Resultado esperado: Se aplica el patrón.	
Evaluación de la prueba: Satisfactoria.	

Tabla 13: Caso de prueba a la historia de usuario 3.

Caso de prueba de Aceptación	
Número: 4.	**Historia de Usuario**: 4.
Nombre: Aplicar Patrón espejo.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción mirror que se encuentra en el boceto en la parte de patrones. 2. Se traza una línea en frente de la entidad, para de esta forma realizar la copia seleccionando la figura y la línea.	
Resultado esperado: Se aplica el patrón.	
Evaluación de la prueba: Satisfactoria.	

Tabla 14: Caso de prueba a la historia de usuario 4.

Caso de prueba de Aceptación	
Número: 5.	**Historia de Usuario**: 5.
Nombre: Aplicar transformación trasladar.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción trasladar que se encuentra en el boceto en la parte de modificaciones. 2. Se selecciona la entidad y se desplaza en el *sketcher* situándola en el lugar que desee.	
Resultado esperado: Se aplica la transformación.	

Evaluación de la prueba: Satisfactoria.

Tabla 15: Caso de prueba a la historia de usuario 5.

Caso de prueba de Aceptación	
Número: 6.	**Historia de Usuario**: 6.
Nombre: Aplicar transformación rotar.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción rotar que se encuentra en el boceto en la parte de modificaciones. 2. Se selecciona un punto dentro del *sketcher* donde a partir de ese punto la entidad rotara.	
Resultado esperado: Se aplica la transformación.	
Evaluación de la prueba: Satisfactoria.	

Tabla 16: Caso de prueba a la historia de usuario 6.

Caso de prueba de Aceptación	
Número: 8.	**Historia de Usuario**: 8.
Nombre: Aplicar transformación escalar.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. El usuario selecciona la opción escalar que se encuentra en el boceto en la parte de modificaciones.	

2. Se selecciona un punto dentro del *sketcher* donde a partir de ese punto la entidad, comenzara a obtener el ángulo deseado por el usuario.
Resultado esperado: Se aplica la transformación.
Evaluación de la prueba: Satisfactoria.

Tabla 17: Caso de prueba a la historia de usuario 7

Caso de prueba de Aceptación	
Número: 8.	**Historia de Usuario**: 8.
Nombre: Aplicar restricción fijo.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	
Entradas/Pasos de ejecución: 1. Se selecciona la entidad. 2. El usuario selecciona el icono que representa la restricción de fijo que se encuentra en el boceto en la parte de *Constraint*.	
Resultado esperado: Se aplica la restricción.	
Evaluación de la prueba: Satisfactoria.	

Tabla 18: Caso de prueba a la historia de usuario 8.

Caso de prueba de Aceptación	
Número: 9.	**Historia de Usuario**: 9.
Nombre: Aplicar restricción coincidencia.	
Condiciones de ejecución: Dibujada en el *sketcher* la curva BSpline.	

Entradas/Pasos de ejecución: 1. Se selecciona la entidad. 2. El usuario selecciona el icono que representa la restricción de coincidencia que se encuentra en el boceto en la parte de *Constraint*.
Resultado esperado: Se aplica la restricción.
Evaluación de la prueba: No Satisfactoria.

Tabla 19: Caso de prueba a la historia de usuario 9.

4.1.1 Sumario de evaluación de las pruebas

Para la ejecución de las pruebas de aceptación se contó con un total de 9 requisitos funcionales. Con la realización de las mismas, se cubrió el 100 % de dichos requisitos. Para la aplicación de estas pruebas se realizaron 2 iteraciones. En la primera iteración se diseñaron 9 casos de pruebas de los cuales 8 fueron satisfactorios dando lugar a 1 no conformidad, la cual fue resuelta, dando lugar a una segunda iteración de 9 casos de pruebas diseñados, de los cuales todos fueron satisfactorios, de esta manera quedó validada la aplicación con un 100% de aceptación.

4.1.2 Cobertura de las pruebas

Para obtener la cobertura de las pruebas de Aceptación que fueron efectuadas en la herramienta se recogieron los siguientes resultados:

1. **Primera iteración**

- Casos de prueba diseñados: 9
- Casos de prueba aplicados: 9
- Cobertura de pruebas (ejecutadas) = 100%

- Cobertura de pruebas (exitosas)= 8/9 = 88%

2. Segunda iteración

- Casos de prueba diseñados: 9
- Casos de prueba aplicados: 9
- Cobertura de pruebas (ejecutadas) = 100%
- Cobertura de pruebas (exitosas)= 9/9= 100%

4.1.3 Consideraciones finales

El proceso de implementación ha satisfecho las necesidades de las tareas pertinentes para darle cumplimiento a los requisitos funcionales de la herramienta AsiXMec, lo cual fue comprobado mediante las pruebas realizadas al sistema dando lugar a un correcto funcionamiento de la misma. Con el fin de este capítulo se da por terminada la propuesta que trae consigo este trabajo.

Conclusiones

Al término de la presente investigación sobre entidad BSpline para la herramienta AsiXMec se arriba a las siguientes conclusiones:

- A partir de la integración a la arquitectura de AsiXMec, la entidad BSpline obtenida cumple con las características del resto de las entidades, lo que permite que sobre ella puedan aplicarse las principales operaciones sobre entidades 2D.
- La existencia de la entidad BSpline genera la posibilidad dentro de AsiXMec de crear diseños que hasta el momento no eran realizables por no contar con una entidad que simulara una curva abierta no uniforme.
- Las pruebas realizadas a las funcionalidades implementadas corroboran que la entidad BSpline creada es una solución adecuada a la problemática que generaba su no existencia.

Recomendación

Se recomienda seguir trabajando sobre la herramienta AsiXMec, específicamente que sobre la entidad BSpline se mantenga un estudio profundo para aumentar las funcionalidades que esta pueda brindarle al usuario, en aras de lograr un producto que sea capaz de brindarle a sus usuarios, funcionalidades de última generación entre los sistemas cad.

Referencias

1. **BLANCO FERNANDEZ, JULIO y SANZ ADAN, FELIX .** *CAD.CAM: GRAFICOS, ANIMACION Y SIMULACION POR COMPUTADOR.* s.l. : PARANINFO, 2002.
2. **JOVER, JOSE MANUEL NAVARRO.** *DISEÑO ASISTIDO POR ORDENADOR CON AUTOCAD.* s.l. : UNIVERSIDAD POLITECNICA DE VALENCIA. SERVICIO DE PUBLICACION, 2006.
3. **luna., Nancy Martínez.** *Curvas y superficies splines.* oaxaca : s.n., 2015.
4. **Quarteroni , Alfio , Sacco, Riccardo y Saleri, Fausto .** *Numerical Mathematics.* Alemania : s.n., 2000.
5. **Carmona, Rhadamés.** *Introducción a la Visualización 3D.* 2008.
6. **álvarez, Núria, Enrich, Carlos y Riera, Luis.** *EL gran libro de 3ds Max.* 2010.
7. **ROLAND HESS, R.** *BLENDER (DISEÑO Y CREATIVIDAD).* s.l. : ANAYA MULTIMEDIA, 2011.
8. **Benítez Herrera, Alejandro y Saurin Ojeda, Gelson.** *Modulo para el comportamiento autónomo de autos en un Entorno virtual urbano.* Ciudad de la Habana : s.n., 2008.
9. **Estrada, José Ángel Lores.** *Módulo de transformación a entidades 2d.* Ciudad de la Habana : s.n., 2012.
10. **González Abad, Israeldis.** *Diseño e Implementación del sistema de Gestión de Información de los recursos de la facultad 3.* Ciudad de la habana : s.n., 2008.
11. **Albán, Oscar Andrés Vivas.** *INTRODUCCIÓN Al QT y Al QT Creator .* UNIVERSIDAD del Cauca : s.n.
12. **Ezust , Alan y Ezust, Paul .** *An Introduction to Design Patterns in C++ with Qt.* 2006.
13. **Sánchez, Tamara Rodríguez.** *Metodología de desarrollo para la Actividad productiva de la UCI.*
14. **Pressman, Roger S.** *"Ingeniería de software. Un enfoque práctico". 6ta Edición. .* 2007. .
15. **Y Sommerville, I.** *"Ingeniería de Software".* 2005. .
16. **L. Bass, P. Clements, R. Kazman,.** *Software Architecture in Practice.* 2003.
17. **(Pérez Mariñán, P.** *Patrones de Diseño (Design Patterns).*
18. **Gómez Argüello, Wilson Javier.** *Metodología de desarrollo de software un enfoque práctico y global versión 1.0.11.*
19. **thelin, Johan.** *Foundations of Qt Development.* 2007.
20. **Salazar, Miguel Arcia.** *Desarrollo de un componente visual para Qt utilizando el framework OpenCascade. .* habana : s.n., 2011.
21. **L. Bass, P. Clements, R. Kazman.** *Software Architecture in Practice.* s.l. : 2nd Edition, 2003.
22. **González, S. G.** *Dibujo asistido con ordenador: teoría y prácticas de diseño con Solidworks .* 2004.
23. **Cellier, F.E.** *Continuous System Modeling.* New York : s.n., 1991.
24. **Booch, Grady, Rumbaugh, James y Jacobson, Ivar.** *El Lenguaje Unificado de Modelado.*
25. **Alvarez, Jesus Carnicer.** *Funciones spline y Diseno Geometrico Asistido por Ordenador.* 2012.
26. **Sánchez, Tamara Rodríguez.** *Metodología de desarrollo para la Actividad productiva de la UCI. .* Ciudad de la habana : s.n., 2015.