Додаток А

Лістинг

```
    /// <summary>
    /// This class renders individual geometry features to a graphics object
using the settings of a map object.
    /// </summary>
    internal static class SharpDXVectorRenderer
    {
        internal const float ExtremeValueLimit = 1E+8f;
        internal const float NearZero = 1E-30f; // 1/Infinity


        internal static readonly GDI.Bitmap DefaultSymbol =
            (GDI.Bitmap)
                GDI.Image.FromStream(
                    Assembly.GetAssembly(typeof (Map))
                        .GetManifestResourceStream("SharpMap.Styles.DefaultSymbol
.png"));


        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawMultiLineString(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, IMultiLineString lines, D2D1.Brush pen, float penWidth,
D2D1.StrokeStyle penStrokeStyle, Map map, float offset)
        {
            for (var i = 0; i < lines.NumGeometries; i++)
            {
                var line = (ILineString)lines[i];
                DrawLineString(renderTarget, factory, line, pen, penWidth,
penStrokeStyle, map, offset);
            }
        }

        internal static Vector2[] OffsetRight(Vector2[] points, float offset)
        {
            int length = points.Length;
            var newPoints = new Vector2[(length - 1) * 2];

            float space = (offset * offset / 4) + 1;

            //if there are two or more points
            if (length >= 2)
            {
                var counter = 0;
                float x = 0, y = 0;
                for (var i = 0; i < length - 1; i++)
                {
                    var b = -(points[i + 1].X - points[i].X);
                    if (b != 0)
                    {
                        var a = points[i + 1].Y - points[i].Y;
                        var c = a / b;
                        y = 2 * (float)Math.Sqrt(space / (c * c + 1));
                        y = b < 0 ? y : -y;
                        x = (c * y);

                        if (offset < 0)
                        {
                            y = -y;
                            x = -x;
                        }
```

```
  newPoints[counter] = new Vector2(points[i].X + x, points[i].Y + y);
   newPoints[counter + 1] = new Vector2(points[i + 1].X + x, points[i + 1].Y +
y);
                    }
                    else
                    {
  newPoints[counter] = new Vector2(points[i].X + x, points[i].Y + y);
   newPoints[counter + 1] = new Vector2(points[i + 1].X + x, points[i + 1].Y +
y);
                    }
                    counter += 2;
                }

                return newPoints;
            }
            return points;
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawLineString(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, ILineString line, D2D1.Brush pen, float penWidth,
D2D1.StrokeStyle penStrokeStyle, Map map)
        {
            DrawLineString(renderTarget, factory, line, pen, penWidth,
penStrokeStyle, map, 0);
        }

        public static void DrawLineString(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, ILineString line, D2D1.Brush pen, float penWidth,
D2D1.StrokeStyle penStrokeStyle, Map map, float offset)
        {
            var points = TransformToImage(line, map);
            if (points.Length > 1)
            {
                using (var geom = new D2D1.PathGeometry(factory))
                {

                    using (var gs = geom.Open())
                    {
                        gs.BeginFigure(points[0], D2D1.FigureBegin.Filled);
                        gs.AddLines(points);
                        gs.EndFigure(D2D1.FigureEnd.Open);

                        gs.Close();
                    }

                renderTarget.DrawGeometry(geom, pen, penWidth, penStrokeStyle);
                }
            }
        }

        private static Vector2 TransformToImage(Coordinate p, Map map)
        {
            //if (map.MapTransform != null && !map.MapTransform.IsIdentity)
            map.MapTransform.TransformPoints(new System.Drawing.PointF[] { p });
            if (p.IsEmpty())
                return new Vector2(float.NaN);

            var height = (map.Zoom * map.Size.Height) / map.Size.Width;
            var left = map.Center.X - map.Zoom * 0.5;
            var top = map.Center.Y + height * 0.5 * map.PixelAspectRatio;

            var x = (float)((p.X - left) / map.PixelWidth);
            var y = (float)((top - p.Y) / map.PixelHeight);
```

```
            if (double.IsNaN(x) || double.IsNaN(y))
                return new Vector2(float.NaN);

            return new Vector2(x, y);
        }

        private static Vector2[] TransformToImage(ILineString line, Map map)
        {
            var height = (map.Zoom * map.Size.Height) / map.Size.Width;
            var left = map.Center.X - map.Zoom * 0.5;
            var top = map.Center.Y + height * 0.5 * map.PixelAspectRatio;


            var cs = line.CoordinateSequence;
            var res = new Vector2[cs.Count];
            for (var i = 0; i < cs.Count; i++)
            {
                var p = cs.GetCoordinate(i);
                var x = (float)((p.X - left) / map.PixelWidth);
                var y = (float)((top - p.Y) / map.PixelHeight);
                res [i] = new Vector2(x, y);
            }
            return res;
        }
        private static Vector2 TransformToImage(ILineString line, Map map, out
Vector2[] points)
        {
            var height = (map.Zoom * map.Size.Height) / map.Size.Width;
            var left = map.Center.X - map.Zoom * 0.5;
            var top = map.Center.Y + height * 0.5 * map.PixelAspectRatio;

            var cs = line.CoordinateSequence;

            var p = cs.GetCoordinate(0);
            var x = (float)((p.X - left) / map.PixelWidth);
            var y = (float)((top - p.Y) / map.PixelHeight);
            var res = new Vector2(x, y);

            points = new Vector2[cs.Count - 1];
            for (var i = 1; i < cs.Count; i++)
            {
                p = cs.GetCoordinate(i);
                x = (float)((p.X - left) / map.PixelWidth);
                y = (float)((top - p.Y) / map.PixelHeight);
                points[i-1] = new Vector2(x, y);
            }
            return res;
        }


        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawMultiPolygon(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, IMultiPolygon pols, D2D1.Brush brush, D2D1.Brush pen, float
penWidth, D2D1.StrokeStyle penStrokeStyle, bool clip, Map map)
        {
            for (var i = 0; i < pols.NumGeometries; i++)
            {
                var p = (IPolygon)pols[i];
                DrawPolygon(renderTarget, factory, p, brush, pen, penWidth,
penStrokeStyle, clip, map);
            }
        }
```

```
        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawPolygon(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, IPolygon pol, D2D1.Brush brush, D2D1.Brush pen, float
penWidth, D2D1.StrokeStyle penStrokeStyle, bool clip, Map map)
        {
            if (pol.ExteriorRing == null)
                return;

            Vector2[] points;
            var startPoint = TransformToImage(pol.ExteriorRing, map, out points);
            if (points.Length > 1)
            {
                using (var geom = new D2D1.PathGeometry(factory))
                {
                    using (var gs = geom.Open())
                    {
                        gs.SetFillMode(D2D1.FillMode.Alternate);

                        gs.BeginFigure(startPoint, D2D1.FigureBegin.Filled);
                        gs.AddLines(points);
                        gs.EndFigure(D2D1.FigureEnd.Closed);

                        for (var i = 0; i < pol.NumInteriorRings; i++)
                        {
                            startPoint =
TransformToImage(pol.GetInteriorRingN(i), map, out points);
                            if (points.Length > 1)
                            {
                                gs.BeginFigure(startPoint,
D2D1.FigureBegin.Filled);
                                gs.AddLines(points);
                                gs.EndFigure(D2D1.FigureEnd.Closed);
                            }
                        }

                        gs.Close();
                    }

                   if (brush != null)
                      renderTarget.FillGeometry(geom, brush);
                   if (pen != null)
                      renderTarget.DrawGeometry(geom, pen, penWidth, penStrokeStyle);
                    }
            }
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawPoint(D2D1.RenderTarget renderTarget, D2D1.Factory
factory, IPoint point, D2D1.Brush b, float size, Vector2 offset, Map map)
        {
            if (point == null)
                return;

            var pp = TransformToImage(point.Coordinate, map);
            if (double.IsNaN(point.X)) return;

            pp += offset;

            var e = new D2D1.Ellipse(pp, size, size);
            renderTarget.FillEllipse(e, b);
            renderTarget.DrawEllipse(e, b);
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
```

```
        public static void DrawPoint(D2D1.RenderTarget renderTarget, D2D1.Factory
factory, IPoint point, D2D1.Bitmap symbol, Vector2 offset,
                                     float rotation, Map map)
        {
            if (point == null)
                return;

            var pp = TransformToImage(point.Coordinate, map);
            if (double.IsNaN(pp.X)) return;
            pp += offset;

            bool symbolCreated = false;
            if (symbol == null) //We have no point style - Use a default symbol
            {
                symbol = CreateDefaultsymbol(renderTarget);
                symbolCreated = true;
            }
            lock (symbol)
            {
                if (rotation != 0 && !Single.IsNaN(rotation))
                {
                    var startingTransform = new
Matrix3x2(renderTarget.Transform.ToArray());

                    var transform = renderTarget.Transform;
                    var rotationCenter = pp;
                    Matrix3x2.Rotation(rotation, rotationCenter);
                    transform *= Matrix3x2.Rotation(rotation, rotationCenter);
                    renderTarget.Transform = transform;

                    var dx = 0.5d*symbol.PixelSize.Width;
                    var dy = 0.5d*symbol.PixelSize.Height;
                    renderTarget.DrawBitmap(symbol, new
SharpDX.RectangleF(Convert.ToSingle(pp.X - dx), Convert.ToSingle(pp.Y + dy),

symbol.PixelSize.Width, symbol.PixelSize.Height),

SharpDX.Direct2D1.BitmapInterpolationMode.Linear);
                    renderTarget.Transform = startingTransform;
                }
                else
                {
                    var dx = 0.5d * symbol.PixelSize.Width;
                    var dy = 0.5d * symbol.PixelSize.Height;
                    renderTarget.DrawBitmap(symbol, new
SharpDX.RectangleF(Convert.ToSingle(pp.X - dx), Convert.ToSingle(pp.Y + dy),

symbol.PixelSize.Width, symbol.PixelSize.Height),

1D2D1.BitmapInterpolationMode.Linear);
                }
            }
            if (symbolCreated)
                symbol.Dispose();
        }

        private static D2D1.Bitmap CreateDefaultsymbol(D2D1.RenderTarget
renderTarget)
        {
            return Converter.ToSharpDXBitmap(renderTarget, DefaultSymbol, 1f);
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
```

```
        public static void DrawMultiPoint(D2D1.RenderTarget renderTarget,
D2D1.Factory factory, IMultiPoint points, D2D1.Bitmap symbol, Vector2 offset,
                                          float rotation, Map map)
        {
            for (var i = 0; i < points.NumGeometries; i++)
            {
           var point = (IPoint)points[i];
           DrawPoint(renderTarget, factory, point, symbol, offset, rotation,
map);
            }
        }

        [MethodImpl(MethodImplOptions.Synchronized)]
        public static void DrawMultiPoint(D2D1.RenderTarget renderTarget,
D2D1.Factory factory,
            IMultiPoint points, D2D1.Brush brush, float size, Vector2 offset, Map
map)
        {
            for (var i = 0; i < points.NumGeometries; i++)
            {
                var point = (IPoint)points[i];
                DrawPoint(renderTarget, factory, point, brush, size, offset,
map);
            }
        }

        private static ClipState DetermineClipState(Vector2[] vertices, int
width, int height)
        {
            float minX = float.MaxValue;
            float minY = float.MaxValue;
            float maxX = float.MinValue;
            float maxY = float.MinValue;


            for (int i = 0; i < vertices.Length; i++)
            {
                minX = Math.Min(minX, vertices[i].X);
                minY = Math.Min(minY, vertices[i].Y);
                maxX = Math.Max(maxX, vertices[i].X);
                maxY = Math.Max(maxY, vertices[i].Y);
            }


            if (maxX < 0) return ClipState.Outside;
            if (maxY < 0) return ClipState.Outside;
            if (minX > width) return ClipState.Outside;
            if (minY > height) return ClipState.Outside;
            if (minX > 0 && maxX < width && minY > 0 && maxY < height) return
ClipState.Within;
            return ClipState.Intersecting;
        }


        private static void DrawPolygonClipped(GraphicsPath gp, PointF[] polygon,
int width, int height)
        {
            var clipState = DetermineClipState(polygon, width, height);
            if (clipState == ClipState.Within)
            {
                gp.AddPolygon(polygon);
            }
            else if (clipState == ClipState.Intersecting)
            {
```

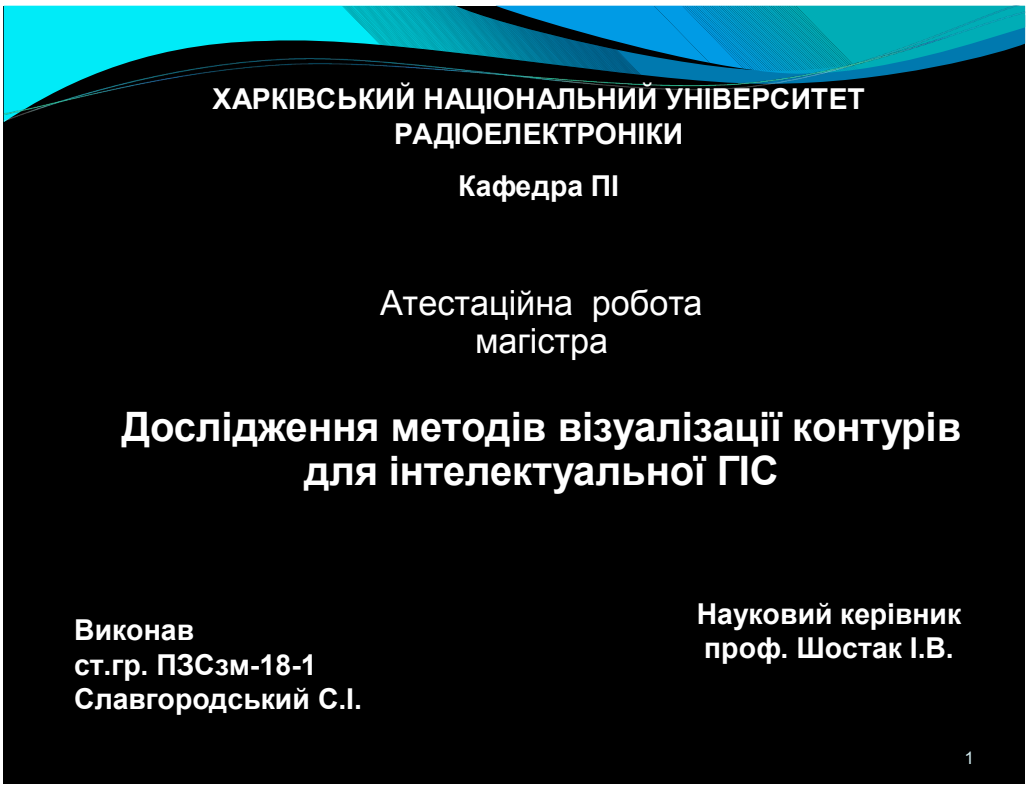
```
                var clippedPolygon = ClipPolygon(polygon, width, height);
                if (clippedPolygon.Length > 2)
                    gp.AddPolygon(clippedPolygon);
            }
        }



        private static Vector2[] LimitValues(Vector2[] vertices, float limit)
        {
            for (var i = 0; i < vertices.Length; i++)
            {
                vertices[i].X = Math.Max(-limit, Math.Min(limit, vertices[i].X));
                vertices[i].Y = Math.Max(-limit, Math.Min(limit, vertices[i].Y));
            }
            return vertices;
        }



        private static Vector2[] ClipPolygon(Vector2[] vertices, int width, int
height)
        {
            var line = new List<Vector2>();
            if (vertices.Length <= 1) /* nothing to clip */
                return vertices;

            for (int i = 0; i < vertices.Length - 1; i++)
            {
                var x1 = vertices[i].X;
                var y1 = vertices[i].Y;
                var x2 = vertices[i + 1].X;
                var y2 = vertices[i + 1].Y;

                var deltax = x2 - x1;
                if (deltax == 0f)
                {
                    // bump off of the vertical
                    deltax = (x1 > 0) ? -NearZero : NearZero;
                }
                var deltay = y2 - y1;
                if (deltay == 0f)
                {
                    // bump off of the horizontal
                    deltay = (y1 > 0) ? -NearZero : NearZero;
                }

                float xin;
                float xout;
                if (deltax > 0)
                {
                    //  points to right
                    xin = 0;
                    xout = width;
                }
                else
                {
                    xin = width;
                    xout = 0;
                }

                float yin;
                float yout;
                if (deltay > 0)
```

```
                {
                    //  points up
                    yin = 0;
                    yout = height;
                }
                else
                {
                    yin = height;
                    yout = 0;
                }

                var tinx = (xin - x1) / deltax;
                var tiny = (yin - y1) / deltay;

                float tin1;
                float tin2;
                if (tinx < tiny)
                {
                    // hits x first
                    tin1 = tinx;
                    tin2 = tiny;
                }
                else
                {
                    // hits y first
                    tin1 = tiny;
                    tin2 = tinx;
                }

                if (1 >= tin1)
                {
                    if (0 < tin1)
                        line.Add(new Vector2(xin, yin));

                    if (1 >= tin2)
                    {
                        var toutx = (xout - x1) / deltax;
                        var touty = (yout - y1) / deltay;

                        var tout = (toutx < touty) ? toutx : touty;

                        if (0 < tin2 || 0 < tout)
                        {
                            if (tin2 <= tout)
                            {
                                if (0 < tin2)
                                {
                                    line.Add(tinx > tiny
                                                ? new Vector2(xin, y1 + tinx *
deltay)
                                                : new Vector2(x1 + tiny *
deltax, yin));
                                }

                         if (1 > tout)
                                {
                                    line.Add(toutx < touty
                                                ? new Vector2(xout, y1 + toutx *
deltay)
                                                : new Vector2(x1 + touty *
deltax, yout));
                                }
                                else
                                    line.Add(new Vector2(x2, y2));
```

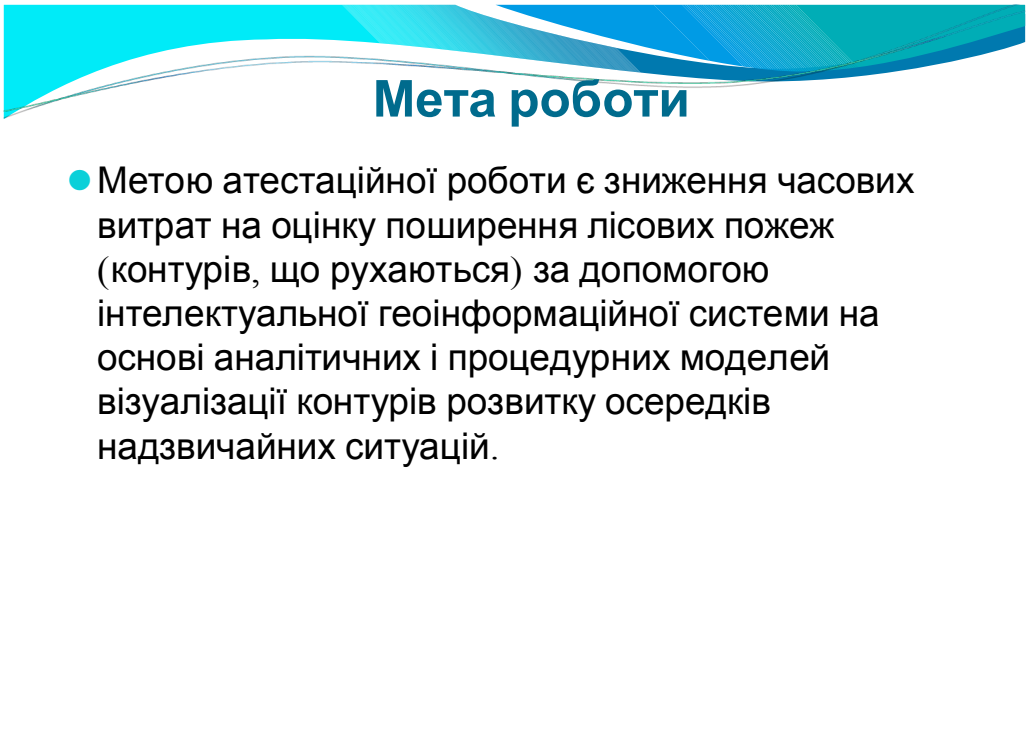
```
                            }
                            else
                            {
                                line.Add(tinx > tiny ? new Vector2(xin, yout) :
new Vector2(xout, yin));
                            }
                        }
                    }
                }
            }
            if (line.Count > 0)
                line.Add(new Vector2(line[0].X, line[0].Y));

            return line.ToArray();
        }

    }
```

Додаток Б

Слайди презентації

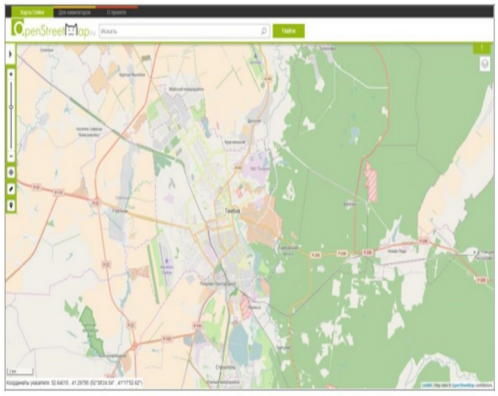


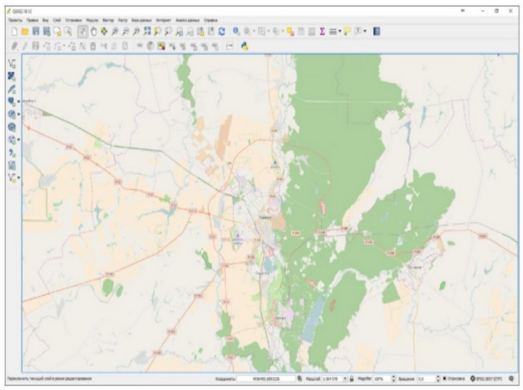
1



2

Аналіз існуючих ГІС

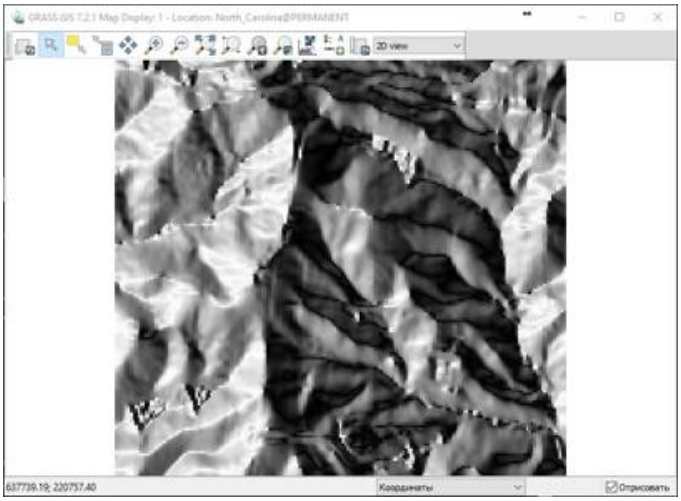




- Вільне програмне забезпечення з відкритим вихідним кодом для побудови геоінформаційних систем, розроблювальне з 2006 року.
- Має низькі системні вимоги й забезпечує кросплатформену роботу.
- Засіб підтримує всі необхідні функції геоінформаційної

3

Інтерфейс візуалізації *GRASS*



- Аналіз показує, що переважна більшість систем не має можливості завантажувати растрові й векторні шари з відкритих джерел, а також моделювати природні й техногенні процеси на місцевості.

4

- Осередки надзвичайних ситуацій, а саме лісові пожежі, що розвиваються в умовах ярусної структури горючих матеріалів, можуть характеризуватися комбінацією типів поширення.

5

Постановка задач дослідження

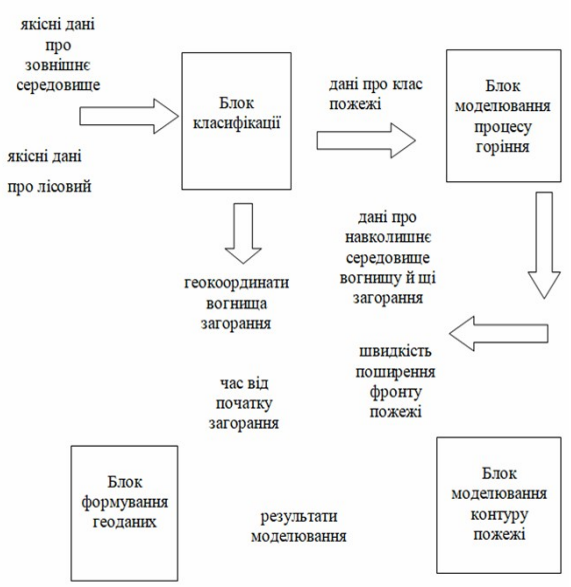
- В атестаційній роботі проведено дослідження аналітичних і процедурних моделей візуалізації контурів розвитку осередків надзвичайних ситуацій, зокрема, лісових пожеж для інтелектуальної геоінформаційної системи підтримки прийняття рішень.

  Для вирішення задачі потрібно провести аналіз:
- аналітичної моделі класифікації пожеж, яка відрізняється нейронечіткою обробкою якісних характеристик навколишнього середовища й вогнища загоряння для класифікації пожеж;
- аналітичних моделей побудови контурів пожеж, які відрізняються використанням якісної інформації про навколишнє середовище (швидкість вітру, напрямок вітру, кут нахилу рельєфу й ін.)
- процедурної моделі побудови й візуалізації контурів лісових пожеж, яка відрізняється згладжуванням підсумкового фронту шляхом наближення правильного багатокутника, описаного навколо контуру, до окружності.
- **Теоретична частина роботи** полягає в застосуванні методів математичного моделювання, що використовують якісні характеристики навколишнього середовища й осередка загоряння при розробці інтелектуальних геоінформаційних систем.
- **Практична частина роботи** полягає в розробці інформаційної технології інтелектуальної геоінформаційної системи для підтримки прийняття рішень, що здійснює моделювання контурів, що рухаються.

6

Структура аналітичної моделі

- використовує якісні характеристики, та містить блок
  - блок класифікації, що одерж якісну інформацію про навколишнє середовище й джерела загоряння, також повертає інформацію про клас пожежі;
  - блок моделювання процесу горіння, що ухвалює дані про клас пожежі й виробляючий розрахунок швидкості поширення фронту пожежі;
  - блок моделювання контуру пожежі, що використовує набір швидкість фронту, що повертає, і точок на поверхн фронту в декартовій системі координат;
  - блок формування геоданих, що одержує результати моделювання контуру пожеж і розраховує геокоординати точок.



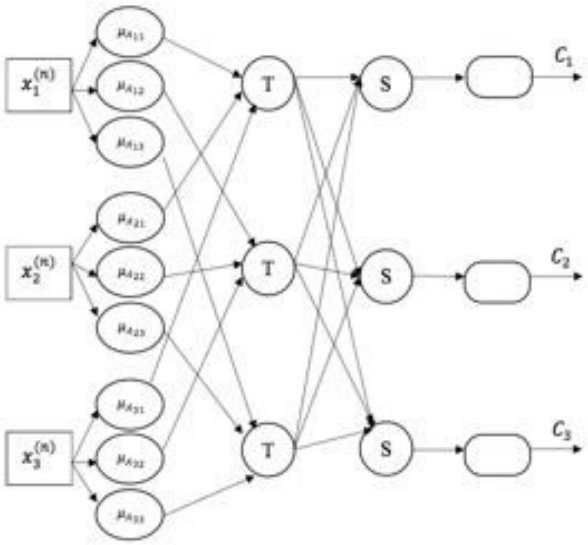
7

Моделі розповсюдження

- Контури пожежі представлено як хвиля, що біжіть. Аналогічно принципу Гюйгенса в оптиці, кожна точка границі пожежі є елементарним джерелом (вогнищем) поширення вогню.
- Горіння з кожної точки границі поширюється в околиці цієї точки у всіх напрямках, де є пальне, причому швидкість поширення в певному напрямку залежить від кута, що утворений даним напрямком з напрямком вітру й (або) схилу.

8

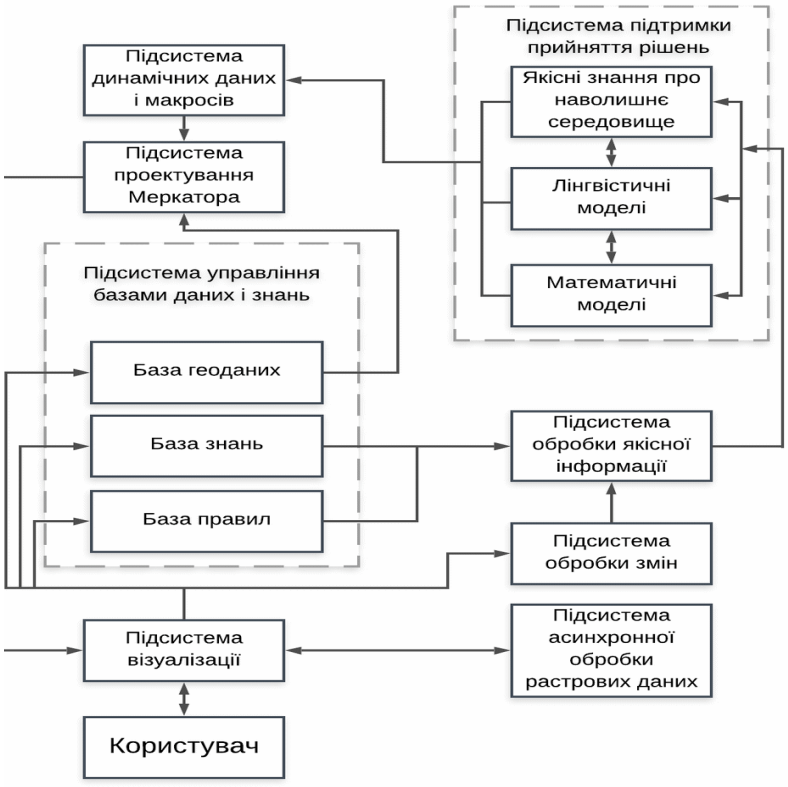
Структура гібридного нейронечіткого класифікатора



- Припущення, що задана множина параметрів навколишнього середовища у вигляді тривимірних чітких векторів, що описують напрямок і швидкість вітру, вологість повітря, які відносяться до трьох класів.
- Правило для заданого вхідного образа формується так, щоб у комбінації нечітких множин кожне з них давало найбільший ступінь приналежності для відповідної вхідної ознаки.
- Якщо ця комбінації неідентична антецедентам уже існуючого правила, то

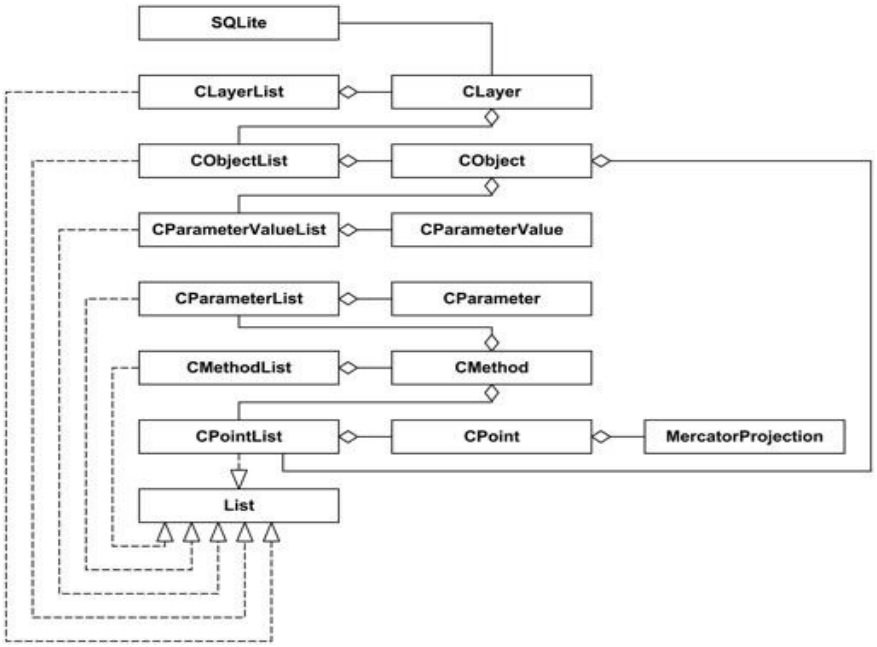
9

UML-модель ІГІС



10

UML-модель взаємодії класів ядра додатку



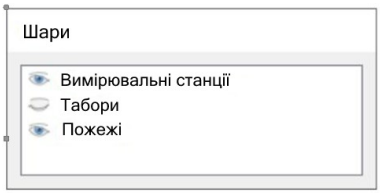
11

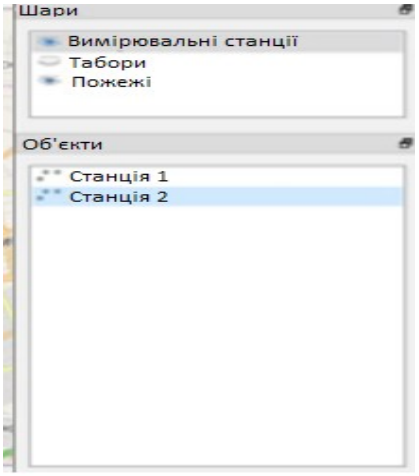
Програмні платформи розробки вузьконаправленої геоінформаційної системи

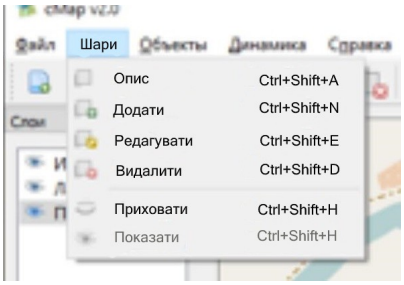
- об'єктно-орієнтований підхід до розробки;
- відносно гнучка робота з базами даних;
- система побудови візуального інтерфейсу;
- вільна програмна платформа;
- висока швидкість роботи й низьке споживання пам'яті;
- кросплатформена розробка.

12

Вікна додатку в режимі відключеного зовнішнього сховища даних

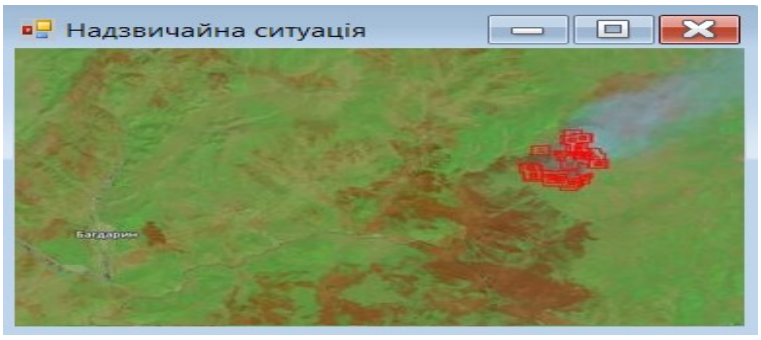






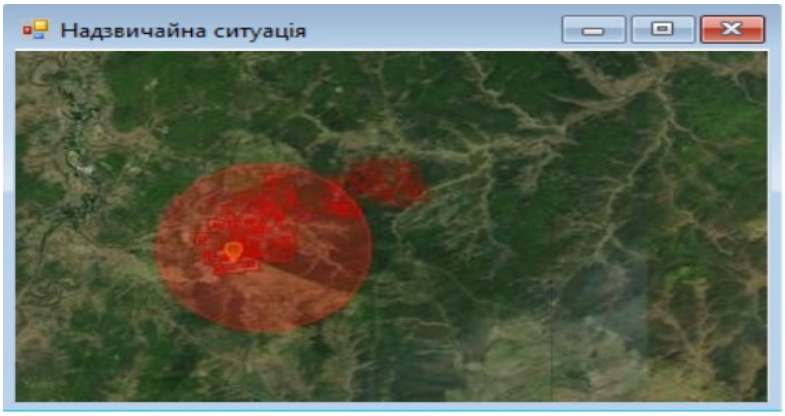
13

Віконна форма “Супутниковий знімок пожежі”



14

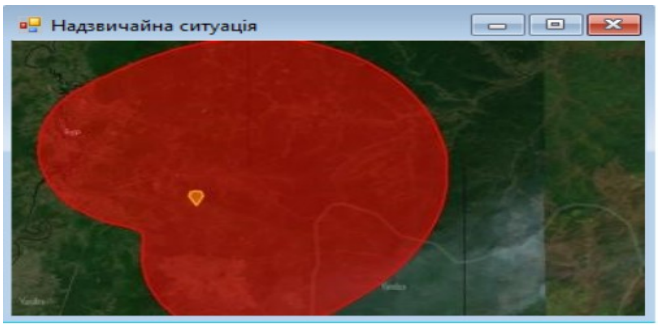
Результати моделювання, що описують контур пожежі через 2 доби після початку загоряння



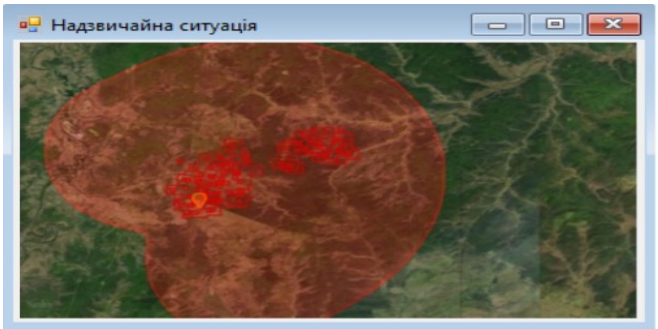
- Таким чином на прикладах з відкритих джерел показано спроможність програмної системи до вирішення поставлених завдань,

15

Результати моделювання



- Модель пожежі з завіхренням



- Сполучення моделі та реального зображення

16

Висновки

- У магістерській роботі вирішене завдання – побудовані аналітична й програмна моделі для інформаційного супроводу інтелектуальної геоінформаційної системи.
- Мета була досягнута – часові витрати скорочені.
- Зроблені імітації деяких моделей.
- Аналіз результатів імітаційних досліджень дає можливість зробити висновок, що розроблена інтелектуальна геоінформаційна система дозволяє одержати географічно точну візуалізацію динаміки руху контурів осередків надзвичайної ситуації, що представлені у вигляді математичних моделей руху на площині.
- Подальший розвиток інформаційної системи може бути пов'язаний з динамічним збором метеоданих контрольних точок, а також аналізом інфрачервоних аерокосмічних знімків реального часу.

УДК 007

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ МОДЕЛЮВАННЯ ТА ВІДОБРАЖЕННЯ ЗОН УРАЖЕННЯ

О.В. Галак, канд. техн. наук, завідувач кафедри
І.Ю. Шубін, канд. техн. наук, професор
С.І. Славгородський, магістрант

У практичних завданнях сучасного етапу розвитку сил і засобів РХБ часто виникає проблема обробки й графічного відображення даних, отриманих у результаті спостережень тривимірних об'єктів – полів і поверхонь. Для рішення актуального завдання моделювання радіохімічної та екологічної обстановки в Україні проведений аналіз проблеми, на підставі якого сформульовані й вирішені основні завдання дослідження. Показано, що для вирішення завдань моделювання екологічної обстановки необхідне створення інформаційної системи, що поєднує геоінформаційну систему з розподіленою базою даних територіальної інформаційно-аналітичної системи екологічного моніторингу, що забезпечує підвищення ефективності збору даних і прогнозування екологічної обстановки, а також забезпечує відображення результатів рішення безпосередньо на карті місцевості. На цій основі запропонований експериментальний варіант інформаційної технології, що використовується для завдання вихідних даних при моделюванні обстановки й відображення результатів рішення на карту України.

У даній роботі розроблені алгоритми кусочно-лінійної інтерполяції однозначної поверхні й побудови ізоліній і розрізів цієї поверхні. Для підвищення якості зображень кусочно-лінійну, що інтерполює поверхню необхідно згладити. Для реальних завдань, пов'язаних з визначенням області на місцевості, цілком задовільними виявляються зображення, побудовані безпосередньо по трикутній сітці. Ізолініями й розрізами вихідної поверхні можна вважати відповідні лінії поверхні, що інтерполює (ламані). Останні повністю визначаються наборами вузлових точок – точок перетинання горизонтальних і вертикальних площин з ребрами багатогранника. Координати вузлових точок обчислюються за допомогою лінійної інтерполяції. З'єднання отриманих вузлових точок гладкої кривої – це найпростіший спосіб зображення гладких ліній. Цей метод не дає гарантії, що при частому завданні рівнів на кресленні не перетнуться різні ізолінії, тому бажане використовувати криву, що інтерполює, близьку за формою до ламаної.

Завдання про перевірку приналежності точки деякої області, виникають, наприклад, коли необхідно перевірити, що деякий населений пункт попадає (або не попадає) у зону впливу факторів поразки, якщо ця зона задана деякою областю. Відповідно до прийнятої полігональної моделі місцевості вважаємо, що всяка область задається замкненою ламаною (простої ламаної), що не має само-перетинів. При цьому вважаємо, що розглянуті планарні підрозбиви є такими, що після видалення із площини, границі областей, що залишаються, є зв'язні. Це потрібно для того, щоб пари точок, що належать однієї області, можна було з'єднати кривою, що не містить граничних точок області.

Часто цілком задовільними виявляються зображення, побудовані безпосередньо по трикутній сітці. Ізолініями й розрізами вихідної поверхні можна вважати відповідні лінії поверхні, що інтерполює (ламані). Останні повністю визначаються наборами вузлових точок – точок перетинання горизонтальних і вертикальних площин з ребрами багатогранника. Координати вузлових точок легко обчислюються за допомогою лінійної інтерполяції.

Програмний комплекс, розроблений на основі викладених алгоритмів, призначений для інтерактивної роботи в режимі реального часу й забезпечує відображення даних, виконання розрахунків і відображення результатів з точністю до хвилин дуги. Комплекс дозволяє виконувати наступні дії: відображати, вводити й редагувати інформацію про населені пункти й параметри осередків впливу; наносити на карту населені пункти й осередкі впливу; будувати області впливу й апроксимуючий багатокутник для довільного набору осередків впливу, відображати їх на карті разом з полігональними лініями рівня, задавати границі області відображення і зони, що заборонені для відображення. Усі основні дані й результати обробки, для полегшення їх сприйняття й підвищення якості аналізу, відображаються в графічному й у текстовому виді. Програмний комплекс відкритий для розширення з боку користувачів.

Додаток Г

Відгук і рецензії

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
ННЦ ЗФН
ВІДЗИВ
на атестаційну роботу магістра

Славгородського Станіслава Ігоровича, гр. ПЗСзм 18-1

Тема: ««Дослідження методів візуалізації контурів для інтелектуальної ГІС»»

спеціальність – **121- Інженерія програмного забезпечення**
Освітня програма – **Інженерія програмного забезпечення**

Аналіз і розробка моделей і методів обробки геометричної інформації, що узагальнюють відомі підходи і методи рішення завдань інтерполяції й дифузії з метою екологічного моніторингу на основі використання ГІС і територіально розподілених баз даних про екологічний стан регіонів, представляє актуальне наукове завдання, що має більше суспільно-соціальну значимість для України. Рішення цього завдання в даній роботі дається для випадку нерегулярних і регулярних сіткових моделей поля забруднення.

Комбінація обробки даних та аналітичного розрахунку – це допомога, яка значно підвищить рівень екологічного моніторингу та визначить шлях подальшого розвитку ситуаціних центрів в Україні.

Для оцінювання екологічної обстановки в будь-якій точці України запропоновані методи моделювання, засновані на інтерполяції даних екологічного моніторингу й прогнозуванні розподілу інтенсивності окремих викидів на основі конуса поширення забруднюючого речовини, що представляє рішення рівняння дифузії. Запропоновані методи дозволяють прогнозувати рівні забруднення середовища на основі методів двовимірної інтерполяції – залежно від завдання, вони забезпечують моделювання екологічної обстановки локально й глобально (для всієї території України) — за рахунок можливості лінійної інтерполяції за значеннями рівнів забруднення в окремих місцях.

Магістрант гр. *ПЗСзм-18-1* Славгородський Станіслав Ігорович готовий до самостійної інженерної діяльності. Атестаційну роботу можна подати до захисту в ЕК за спеціальністю *121 – «Інженерія програмного забезпечення»*, освітньо-професійною програмою *Програмне забезпечення систем.*

«16» 12 2019р.

Керівник атестаційної роботи магістра
І.В. Шостак, д.т.н., проф. каф. ПІ

UNICHECK

Власник документу:
Нечволод Вадим Юрійович каф. ПІ

ID перевірки:
1000752711

Дата перевірки:
10.12.2019 12:25:29 GMT+0

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.12.2019 14:26:04 GMT+0

ID користувача:
94949

Назва документу: 2019_ПІ_ПЗСзм-18-1_Славгородський С.І. _скорочений

ID файлу: 1000764606 Кількість сторінок: 33 Кількість слів: 5625 Кількість символів: 43963 Розмір файлу: 869.43 KB

3.08% Схожість

Найбільша схожість: 2.03% з джерело http://vesnat.ru/nuda/institut-kompyuternih-tehnologij/stranica-4.html

3.08% Схожість з Інтернет джерелами 24 ..... Page 35

0.16% Текстові збіги по Бібліотеці акаунту 1 ..... Page 35

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів 6

Страница 1 из 35 — +

Рецензія
на атестаційну роботу магістра
магістранта групи ПЗСзм-18-1 Славгородського Станіслава Ігоровича
спеціальність – 121- Інженерія програмного забезпечення
освітньо-професійна програма Програмне забезпечення систем

«Дослідження методів візуалізації контурів для інтелектуальної ГІС».

Структура атестаційної роботи: пояснювальна записка _____стор.; графічна частина **17** аркушів; програмне застосування (прикладна програма) __12_файлів загальним обсягом _1200_ Кбайт.

Актуальність теми атестаційної роботи визначається тим, що розробленіалгоритми дозволяють розширити можливості розрахунків при побудові границь зі складною формою. Описані алгоритми використовують сучасні методи та теореми дляшвидкодії обчислень. Описані математичні моделі дозволяють застосуватидодаткові параметри для максимально вірних результатів.У роботі розглянуто актуальну на сьогодні проблематику та питання оптимізації.

Зміст роботи відповідає завданню та темі атестаційної роботи. Обсяг записки та домірність окремих розділів відповідає вимогам до атестаційної роботи магістранта.

В атестаційній роботі оброблено досить велику кількість наукового матеріалу, на високому теоретичному та методологічному рівні проведено дослідження особливостей і методів обробки та покращення обчислень при виділенні та побудові границь територій.

Запропоновані моделі та алгоритми роботи добре обґрунтовані. Теоретичні і практичні результати мають достатньо високий науково-технічний рівень та якість розробки базується на відомих і перспективних технологіях, студент показав здатність формулювати власну точку зору на основі аналізу думок різних вчених в цій галузі.

Пояснювальну записку до атестаційної роботи викладено з дотриманням внутрішньої логіки, між розділами простежується взаємозв'язок.

До недоліків атестаційної роботи слід віднести те, що деяким ключовим аспектам обчислень приділено мало уваги та деякі математичні формули були опущені як загальновідомі.

Атестаційна робота магістранта групи ПЗСзм-18-1 Славгородського Станіслава Ігоровича. відповідає вимогам до атестаційних робіт і заслуговує оцінки «добре 80». Атестаційну роботу можна представити для захисту в ЕК за спеціальністю 121 - Інженерія програмного забезпечення, освітньо-професійною програмою Програмне забезпечення систем.

Рецензент,
к.т.н, проф. каф. ІІІ  І.Ю. Шубін

Рецензія
на атестаційну роботу магістра
магістранта групи **ПЗСм-18-1** ***Славгородського Станіслава Ігоровича***
спеціальність – **121- Інженерія програмного забезпечення**
Освітня програма **Програмне забезпечення систем**

Тема атестаційної роботи: «Дослідження методів візуалізації контурів для інтелектуальної ГІС»

Структура атестаційної роботи: пояснювальна записка _____стор.; графічна частина 12 аркушів; програмне застосування (прикладна програма) 12 файлів загальним обсягом 1200Кбайт.

Атестаційної робота, що рецензується, присвячена створенню моделей побудови та обробки границь зі складною формою із застосуванням розрахунків кривих ліній на окремих ділянках.

Актуальність теми атестаційної роботи визначається тим, що розглядаються існуючи проблеми автоматизації моніторингу навколишнього середовища й забезпечення можливості їх рішення на основі створених математичних моделей та алгоритмів контролю та прогнозуванню надзвичайних ситуацій. Описані алгоритми та методи розроблені на основі сучасних теорем та припущень, особливу увагу приділено оптимізації обробки та обчислень.

Зміст роботи відповідає завданню та темі атестаційної роботи. Атестаційна робота виконана у відповідності до завдання та чинних вимог з дотриманням внутрішньої логіки, самостійно та в повному обсязі.

В атестаційній роботі студент Славгородський С.І. детально проаналізував існуючі можливості сучасних алгоритмів та методів, ознайомився з відомими науковими роботами та привів аргументовані думки із застосуванням ілюстрацій прикладів та формул при побудові власної математичної моделі для покращення обчислень та оптимізації у розрахунках границь зі складною формою.

Магістрант показав здатність формулювати власну думку, вміння чітко та зрозуміло проводити аналіз теоретичних матеріалів та на їх основі впроваджувати власні моделі та алгоритми. Описані розрахунки мають вагомі теоретичні і практичні результати, що свідчить про достатньо високий науково-технічний рівень. Як результат була розроблена система обчислень складних границь.

До недоліків атестаційної роботи слід віднести те, що представлення математичних методів та моделі місцями порушується відповідно до загальноприйнятих правил.

Атестаційна робота магістра гр. ПЗСм-18-1 Славгородського Станіслава Ігоровича відповідає вимогам до атестаційних робіт магістрів і заслуговує оцінки «добре (85)».

Атестаційну роботу можна представити для захисту в ЕК за спеціальністю *121-Інженерія програмного забезпечення*, освітньо-професійною програмою *Програмне забезпечення систем*

Рецензент:
д.т.н., професор,
завідувач кафедри ІІІ В.О. Філатов