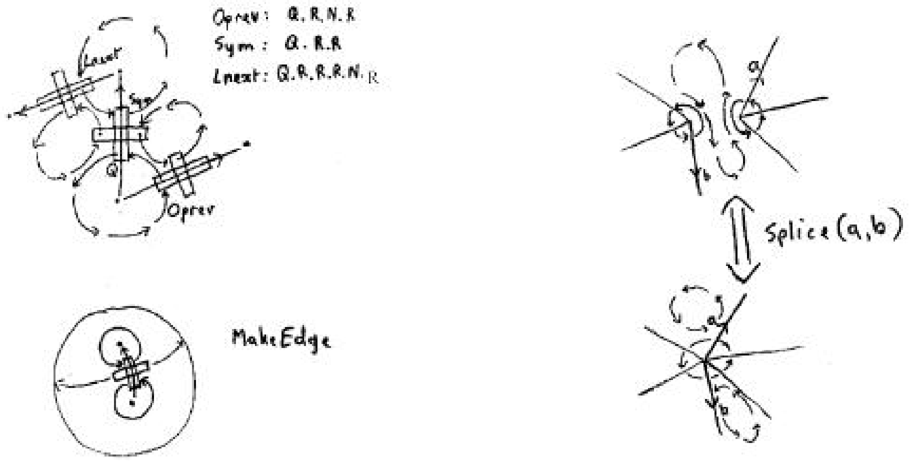
Table 1 —Activity, employment, and unemployment rates for ages 15-64, by sex and urban/rural location, 1990–95

					Years			
Rate			1990	1991	1992	1993	1994	1995
Activity rate (15-64)**								
	Urban	Male	72.0	70.8	69.8	70.8	70.5	69.9
		Female	22.4	20.3	19.1	20.3	20.2	19.5
		Total	47.3	45.7	44.4	45.7	45.4	44.8
	Rural	Male	78.2	76.8	76.3	76.3	76.1	76.3
		Female	34.7	29.3	26.1	24.0	25.4	23.0
		Total	56.4	53.0	50.8	50.2	51.0	49.7
	Total	Male	75.3	74.0	73.2	73.7	73.5	73.3
		Female	29.0	25.2	22.8	22.3	23.0	21.4
		Total	52.1	49.6	47.8	48.1	48.4	47.4
Employment rate (15-64)**								
	Urban	Male	67.0	65.3	64.7	64.9	64.9	64.6
		Female	16.8	15.4	14.3	14.6	14.6	14.1
		Total	42.1	40.5	39.5	39.9	39.8	39.4
	Rural	Male	74.5	72.4	71.8	70.9	70.7	70.6
		Female	31.6	26.2	22.8	19.5	20.4	18.1
		Total	53.0	49.3	46.9	45.3	45.8	44.3
	Total	Male	71.0	69.1	68.5	68.1	68.0	67.8
		Female	24.7	21.1	18.9	17.2	17.7	16.2
		Total	47.9	45.2	43.4	42.8	43.0	42.0
Unemployment Rate (15-64)**								
	Urban	Male	6.9	7.7	7.3	8.4	7.9	7.6
		Female	24.8	24.4	24.9	27.9	28.0	27.6
		Total	11.1	11.4	11.1	12.7	12.4	11.9
	Rural	Male	4.7	5.7	5.9	7.0	7.1	7.5
		Female	9.0	10.8	12.5	18.7	19.6	21.4
		Total	6.0	7.1	7.6	9.8	10.1	10.7
	Total	Male	5.7	6.6	6.5	7.6	7.5	7.5
		Female	14.7	15.9	17.3	22.7	23.1	24.1
		Total	8.2	8.9	9.1	11.1	11.1	11.3

Source: CAPMAS, LFSS.

Notes: Activity rate = labor force/population x 100 percent; employment rate = employment/population x 100 percent; unemployment rate = unemployment/labor force x 100 percent.



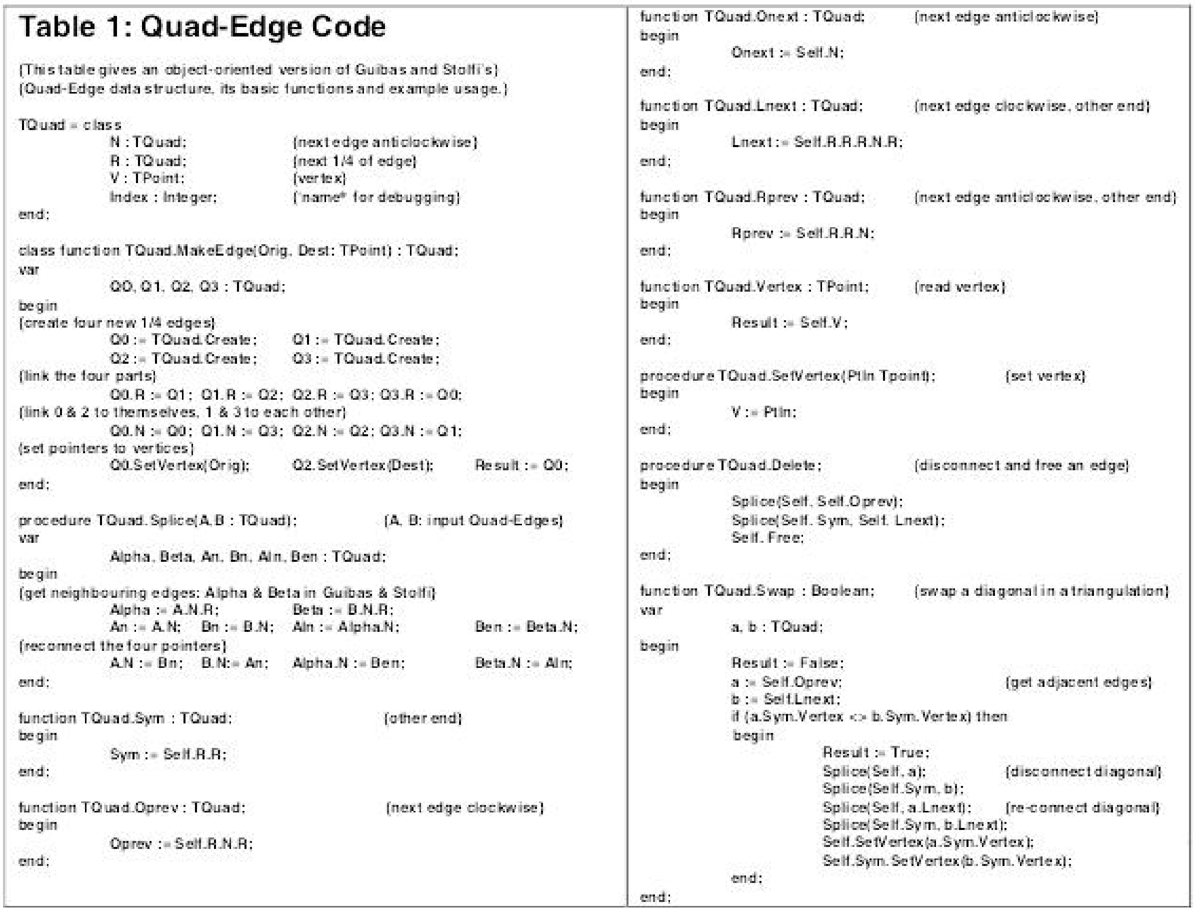


Table 1: Quad-Edge Code

```
{This table gives an object-oriented version of Guibas and Stolfi's}
{Quad-Edge data structure, its basic functions and example usage.}

TQuad = class
            N : TQuad;              {next edge anticlockwise}
            R : TQuad;              {next 1/4 of edge}
            V : TPoint;             {vertex}
            Index : Integer;        {'name' for debugging}
end;

class function TQuad.MakeEdge(Orig, Dest: TPoint) : TQuad;
var
            Q0, Q1, Q2, Q3 : TQuad;
begin
{create four new 1/4 edges}
            Q0 := TQuad.Create;     Q1 := TQuad.Create;
            Q2 := TQuad.Create;     Q3 := TQuad.Create;
{link the four parts}
            Q0.R := Q1;  Q1.R := Q2;  Q2.R := Q3; Q3.R := Q0;
{link 0 & 2 to themselves, 1 & 3 to each other}
            Q0.N := Q0;  Q1.N := Q3;  Q2.N := Q2; Q3.N := Q1;
{set pointers to vertices}
            Q0.SetVertex(Orig);     Q2.SetVertex(Dest);     Result := Q0;
end;

procedure TQuad.Splice(A,B : TQuad);            {A, B: input Quad-Edges}
var
            Alpha, Beta, An, Bn, Aln, Ben : TQuad;
begin
{get neighbouring edges: Alpha & Beta in Guibas & Stolfi}
            Alpha := A.N.R;         Beta := B.N.R;
            An := A.N;   Bn := B.N;   Aln := Alpha.N;       Ben := Beta.N;
{reconnect the four pointers}
            A.N := Bn;   B.N:= An;    Alpha.N := Ben;       Beta.N := Aln;
end;

function TQuad.Sym : TQuad;                     {other end}
begin
            Sym := Self.R.R;
end;

function TQuad.Oprev : TQuad;                   {next edge clockwise}
begin
            Oprev := Self.R.N.R;
end;

function TQuad.Onext : TQuad;           {next edge anticlockwise}
begin
            Onext := Self.N;
end;

function TQuad.Lnext : TQuad;           {next edge clockwise, other end}
begin
            Lnext := Self.R.R.R.N.R;
end;

function TQuad.Rprev : TQuad;           {next edge anticlockwise, other end}
begin
            Rprev := Self.R.R.N;
end;

function TQuad.Vertex : TPoint;         {read vertex}
begin
            Result := Self.V;
end;

procedure TQuad.SetVertex(PtIn Tpoint);             {set vertex}
begin
            V := PtIn;
end;

procedure TQuad.Delete;                 {disconnect and free an edge}
begin
            Splice(Self, Self.Oprev);
            Splice(Self. Sym, Self. Lnext);
            Self. Free;
end;

function TQuad.Swap : Boolean;          {swap a diagonal in a triangulation}
var
            a, b : TQuad;
begin
            Result := False;
            a := Self.Oprev;                            {get adjacent edges}
            b := Self.Lnext;
            if (a.Sym.Vertex <> b.Sym.Vertex) then
            begin
                        Result := True;
                        Splice(Self, a);            {disconnect diagonal}
                        Splice(Self.Sym, b);
                        Splice(Self, a.Lnext);      {re-connect diagonal}
                        Splice(Self.Sym, b.Lnext);
                        Self.SetVertex(a.Sym.Vertex);
                        Self.Sym. SetVertex(b. Sym. Vertex);
            end;
end;
```

To show the simplicity of its use, Fig. 3 shows the two commands that are used to modify a graph: "Make-edge" to create a new edge on a manifold, and "Splice" to connect/disconnect Quad-edges together. In the simplest case, Splice connects two separate "Next" loops, joining the two nodes together, and at the same time splitting the "Next" loop around the common face. (The